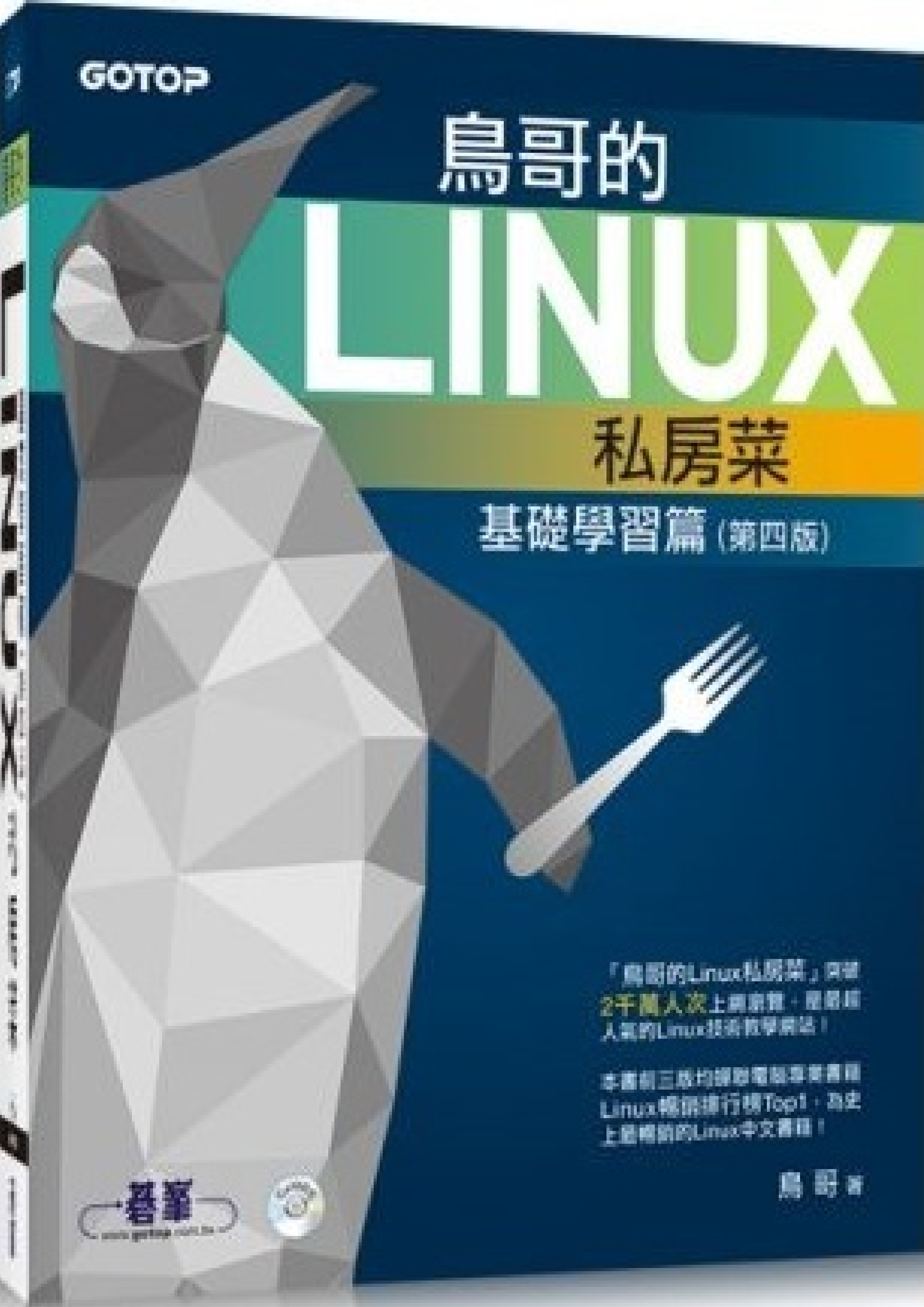

GOTOP
鳥哥的
LINUX
私房菜
基礎學習篇（第四版）
「鳥哥的Linux私房菜」突破
2千萬人次上網瀏覽，是最具
人氣的Linux技術教學網站！
本書前三版均榮膺電腦專業書籍
Linux暢銷排行榜Top1，為史
上最暢銷的Linux中文書籍！
鳥哥 著
碁峰

目錄

鸟哥的Linux私房菜：基础学习篇 第四版

> 来源：鳥哥的 Linux 私房菜 - 基礎學習篇

目录及概述

这本书的所有内容是学习Linux的基础，这些内容是基础中的基础，如果您能将其中的文字都看完并且消化过，那么未来在管理 Linux主机以及架设网站方面，就能够达到“事半功倍”的成效，请不要忽略这些内容了！否则，再怎么讨论都是枉然的啦！^_^。 Linux的资料非常的多，每份资料彼此的相关性都很强，要单独的一项一项讲解并不容易， 那么这本书件该怎么看呢？建议先按照顺序将内容大致浏览过一次，看不懂的地方也可以先略过不要紧。 全部看完之后，再从头开始“仔细”的实际操作过一遍，那应该就能够进入Linux的世界啰～

另外，每个章节下面的日期，指的是重大改版日期而非最新日期，最新日期请以该章节结束部分的工作日志为主的喔！

Linux 的学习曲线，一个老人家的建议！

第一部份 Linux 的规划与安装

常常听到Linux具有非常优良的血统，所以具有相当良好的多用户多任务环境，可以方便程序设计师来开发软件。 此外，Linux本身是不用钱的“自由软件”，使用上面并没有所谓的“盗版”问题。但是，为什么Linux不用钱？ 随便修改或发布Linux为什么不会被罚？为什么Linux有这么多的版本？包括Fedora, SuSE, CentOS, Debian等等？ 这个都是我们必须要来了解的部分！了解这些部分，你才会对Linux有一个正确的理解，才能够跟你的同事、同学、 上司说明，为什么使用Linux具有很多优点与好处！ ^_^

Linux并不好学习，鸟哥也是“重伤”过好几次才能对Linux有一些基础的认知。那么到底应该如何学习Linux呢？关键在实作。 既然要实作就得要实际的安装一部Linux，那么Linux要安装前需要熟悉哪些基础观念？计算机概论是非常重要的一环！ 因为Linux与硬件的关系还不小～此外，打造一台Windows/Linux共存的主机也是很有用的， 至少对于需要多平台但又缺乏空间与金钱的朋友来说，这样的处理是非常有用的！

在第一篇里面，我们会由计算机概论谈起，再讲到Linux的历史渊源与自由软件的关系，然后重点在于如何规划硬件与Linux安装， 最后谈到如何登陆与使用Linux图形/命令行的环境。本篇数据较多，第一次接触Linux的新朋友， 很多数据若看不懂可以先略过，等到后续文章都读完了再回来看，才会有帮助喔！ ^_^

第零章 计算机概论

鸟哥在大专院校的教学经验中发现到，由于对 Linux 有兴趣的朋友很多可能并非信息相关专业出身， 因此对于计算机硬件及计算机方面的概念不熟。然而操作系统这种咚咚跟硬件.....2015/04/16

第一章 Linux是什么/如何学习

众所皆知的，Linux的核心原型是1991年由托瓦兹（Linus Torvalds）写出来的，但是托瓦兹为何可以写出Linux这个操作系统？ 为什么他要选择386的计算机来开发？为什么Linux的发展可以这么迅速？又为什么Linux是免费的？.....2015/04/23

第二章 主机规划与磁盘分区

事实上，要安装好一部Linux主机并不是那么简单的事情，你必须要针对distributions的特性、服务器的软件能力、 未来的升级需求、硬件扩充性需求等等来考虑，还得要知道磁盘分区、文件系统.....2015/04/28

第三章 安装CentOS 7.x与多重开机技巧

Linux distributions越作越成熟，所以在安装方面也越来越简单！虽然安装非常的简单， 但是刚刚前一章所谈到的基础认知还是需要了解的，包括MBR, partition, boot loader, mount, software的.....2015/05/06

第四章 首次登陆与线上求助 man page

终于可以开始使用Linux这个有趣的系统了！由于Linux系统使用了非同步的磁盘/内存数据传输模式， 同时又是个多用户多任务的环境，所以你不能随便的不正常关机，关机有一定的程序喔！错误的关机方法.....2015/06/02

第二部分 Linux 文件、目录与磁盘格式

安装完了 Linux 之后，接着下来自然就是要使用他了！我们在 开机与关机及简易指令操作 稍微说明了指令下达的方法，以及指令线上查询的方式， 因此您可以轻易的使用命令行界面来进行诸多的动作与工作。那么接着下来呢？当然就是想要知道 Linux 里面有什么东西啰，所以，在这一个部分当中，我们将介绍 Linux 最基本的文件权限概念， 与每个文件目录所带有的意涵。

当然啰，要了解权限的概念，那么对于不同的“身份”就需要了解一下才行， 不同的身份的人，所创建的或拥有的文件是否会相同呢？例如系统管理员与一般身份使用者的文件？ 当然不太一样！除此之外，如果您的硬盘空间不足，需要增加硬盘时，应该要如何新增呢？ 还有，内存不足的情况下，有没有增进虚拟内存容量的方法？在接下来的几个章节之中，我们将介绍 Linux 主要的文件架构、以及磁盘在 Linux 当中该如何使用及挂载等问题。

第五章 Linux文件权限与目录配置

Linux最优秀的地方之一，就在于他的多用户多任务的环境。而为了让各个使用者具有较保密的文件数据， 因此文件的权限管理就变的很重要了。 Linux一般将文件可存取的身份分为三个类别，分别是 owner/group/other， 且三种身份各有read/write/execute.....2015/06/03

第六章 Linux文件与目录管理

在第五章我们认识了Linux系统下的文件权限概念以及目录的配置说明。 在这个章节当中，我们就直接来进一步的操作与管理文件与目录吧！包括在不同的目录间变换、 创建与删除目录、创建与删除文件， 还有寻找文件、查阅文件内容.....2015/06/16

第七章 Linux 磁盘与文件系统管理

系统管理员很重要的任务之一就是管理好自己的磁盘文件系统，每个分区不可太大也不能太小，太大会造成磁盘容量的浪费， 太小则会产生文件无法储存的困扰。此外，我们在前面几章谈到的文件权限与属性中， 这些权限与属性分别记录在.....2015/06/26

第八章 文件的压缩与打包

在 Linux 下面有相当多的压缩指令可以运行喔！这些压缩指令可以让我们更方便从网络上面下载大型的文件呢！ 此外，我们知道在 Linux 下面的扩展名是没有什么很特殊的意义的，不过，针对这些压缩指令所做出来的压缩文件， 为了方便记忆.....2015/07/16

第三部分：学习 **Shell 与 Shell scripts**

了解了基本的 Linux 文件属性与目录的配置之后，在进入更深入的 Linux 世界之前，有几个课题还是一定要知道的，那就是我们所使用的这个文字模式接口，也就是所谓的“Shell”这个咚咚。在 Linux 的世界中，使用的是 GNU 发展出来的强化的第二代 shell ，称为 BASH Shell ，他有什么特异功能呢？简单的说，我们之前下达的几个指令都是 bash 管理的，除此之外，他还可以记录指令、文件或命令的补全功能、环境变量的使用等等，还有很多功能等着你去发掘呢！

在知道了部分的 bash 功能后，在接着下来，我们还得了解一下什么是数据流重导向？还有常规表达式等等的问题， 这都是未来我们系统管理员在管理主机上面，一个不可缺乏的利器！当然啰，要将这些功能整合起来运用的话， 就不能不学习一下所谓的脚本“ shell scripts ”，他具有基础的程序能力（ Program ），当真是个管理系统的好帮手呢！

再来，在未来的建站设置当中，常会使用到文字编辑器来编辑参数配置文件，这个时候， 系统管理员至少务必要熟悉一套命令行下的文书编辑软件，当然不限制哪一套软件啦，但是 vi 是最标准的 Unix-Like 的命令行之文书处理软件，所以，我们几乎一定可以在每部 Unix-Like 上面发现他的踪迹，所以，就来了解他一下吧，这也是挺重要的工作呢！

第九章 vim程序编辑器

系统管理员的重要工作就是得要修改与设置某些重要软件的配置文件， 因此至少得要学会一种以上的命令行的文书编辑器。 在所有版本的 Linux 上头都会有的一套文书编辑器就是 vi ，而且很多软件.....2015/07/07

第十章 认识与学习 BASH

在 Linux 的环境下，如果你不懂 bash 是什么，那么其他的东西就不用学了！ 因为前面几章我们使用终端机下达指令的方式， 就是通过 bash 的环境来处理的喔！ 所以说，他很重要吧！bash 的东西非常的多，包括变量.....2015/07/09

第十一章 正则表达式与文件格式化处理

正则表达式 （Regular Expression, RE, 或称为常规表达式）是通过一些特殊字符的排列， 用以“搜寻/取代/删除”一列或多列文字字串， 简单的说， 正则表达式就是用在字串的处理上面的一项“表示式”。正则表达式并.....2015/07/14

第十二章 学习 shell scripts

如果你真的很想要走信息这条路，并且想要好好的管理好属于你的主机，那么，别说鸟哥不告诉你， 可以自动管理你的系统的好工具： Shell scripts 真的是得要好好学习学习的！ 基本上， shell script 有点像是早期的批处理文件，亦即是.....2015/07/17

第四部分：Linux 使用者管理

好了！终于要到了管理 Linux 帐号的时刻了！对于 Linux 有一定的熟悉度之后，再来就是要管理连上 Linux 的帐号问题了！这个帐号的问题可大可小啦！大到可以限制他使用 Linux 主机的各项资源，小到甚至一般帐号的密码订定规则都可以进行规定！端看您对于安全的需求啦！此外，如果站在资源平均分配的角度上，那么 Linux 主机上面有限的资源当然是平均分配给大家比较好！这个时候就得来规定一下“谁可以使用多少的硬盘空间？”那就是 Quota 喔！呵呵！厉害吧！

在订定完了一些帐号的规则之后，那么我们就继续来管理一下主机的系统与程序的管理吧！这个包括了观察每个程序 （Process） 与工作调度及工作管理 （ jobs control ） ，这些也都是很重要的工作呢！

第十三章 Linux 帐号管理与 ACL 权限控制

要登陆 Linux 系统一定要有帐号与密码才行，否则怎么登陆，您说是吧？不过， 不同的使用者应该要拥有不同的权限才行吧？我们还可以通过 user/group 的特殊权限设置，来规范出不同的群组开发专案呢.....2015/07/22

第十四章 磁盘配额（**Quota**）与进阶文件系统管理

如果您的 Linux 服务器有多个用户经常存取数据时，为了维护所有使用者在硬盘容量的公平使用，磁盘配额 （Quota） 就是一项非常有用的工具！另外，如果你的用户常常抱怨磁盘容量不够用，那么更进阶的文件系统就得要学习学习.....2015/07/28

第十五章 例行性工作调度 （crontab）

学习了基础篇也一阵子了，你会发现到为什么系统常常会主动的进行一些任务？ 这些任务到底是谁在设置工作的？ 如果你想要让自己设计的备份程序可以自动的在系统下面执行，.....2015/07/31

第十六章 程序管理与 SELinux 初探

一个程序被载入到内存当中运行，那么在内存内的那个数据就被称为程序（process）。程序是操作系统上非常重要的概念， 所有系统上面跑的数据都会以程序的型态存在。那么系统的程序有哪些状态.....2015/08/08

第五部分：Linux 系统管理员

嗯！终于来到系统管理员 （ root ） 要注意的工作事项之篇幅了！各位准系统管理员心理准备好了吗？ 我们要管理机器啰，呵呵！那么管理员的工作是什么？看报喝茶？！没错！管理员最大的享受就是看报喝茶了。 一个好的系统管理员，平时不会希望挂载网站上面一再不断的查询、检查漏洞等等的， 因为果真如此的话，那么就表示“机器一定有问题了！”。为了让我们的 Linux 机器跑得更稳更顺畅，好让我这个管理员有更多的时间去看报喝茶，哈哈！ 更深入的了解系统是需要的！所以，这一篇我们由开机关机的整体流程谈起，好了解一下 Linux 在开机的过程中到底做了哪些事情，这样才能知道我们在什么时候应该做什么事情呐！

此外，由于“没有一个套件是永远安全的！”，所以套件管理是相当重要的一部份，这里我们以 RPM 与 Tarball 来介绍一下如何管理你系统上面的套件。再来，你知道你的系统上面跑了多少数据吗？ 虽然知道什么是 ps 来查询程序，但是总是得知道我的系统有哪些服务吧！嘿嘿！来看看先?不但如此，还得针对登录文件进行解析，以及对于系统进行备份。呵呵！ 管理员的工作还真多那。不止不止，还要进行核心的管理呢！哇！果然是忙毙了！无论如何， 还是得要了解呐！

第十七章 认识系统服务（daemon）

在 Unix-Like 的系统中，你常常听到 daemon 这个字眼！那么什么是传说中的 daemon 呢？这些 daemon 放在什么地方？他的功能是什么？该如何启动这些 daemon ？又如何有效的将这些 daemon 管理妥当.....2015/08/14

第十八章 认识与分析登录文件

当你的 Linux 系统出现不明原因的问题时，很多人都告诉你，你要查阅一下登录文件才能够知道系统出了什么问题了， 所以说，了解登录文件是很重要的事情呢。登录文件可以记录系统在什么时间、哪个主机、哪个服务.....2015/08/20

第十九章 开机流程、模块管理与 loader

系统开机其实是一项非常复杂的程序，因为核心得要侦测硬件并载入适当的驱动程序后， 接下来则必须要调用程序来准备好系统运行的环境，以让使用者能够顺利的操作整部主机系统。 如果你能够理解开机的原理.....2015/08/31

第二十章 网络设置与备份策略

新的 CentOS 7 有针对不同的服务提供了相当大量的命令行设置模式，因此过去那个 setup 似乎没有什么用了！ 取而代之的是许多加入了 bash-complete 提供了不少参数补全的设置工具！甚至包括网络设置也是通过这个机制哩！ 我们这个小章.....2015/09/03

第二十一章 软件安装：源代码与 Tarball

我们在第一章、Linux是什么当中提到了 GNU 计划与 GPL 授权所产生的自由软件与开放源码等咚咚。 不过，前面的章节都还没有提到真正的开放源码是什么的讯息！在这一章当中，我们将借由 Linux 操作系统里面的可执行文件.....2015/09/06

第二十二章 软件安装：RPM, SRPM 与 YUM 功能

虽然使用源代码进行编译可以进行客制化的设置，但对于 Linux distribution 的原本发布商来说， 则有软件管理不易的问题， 毕竟不是每个人都会进行源代码编译的。 如果能够将软件预先在相同的硬件与操作系统上面编译好才发布的话.....2015/09/09

第二十三章 X Window 设置介绍

在 Linux 上头的图形接口我们称之为 X Window System，简称为 X 或 X11 啰！为何称之为系统呢？这是因为 X 窗口系统又分为 X server 与 X client ，既然是 Server/Client （主从架构）这就表示其实 X 窗口系统是可以跨网络且跨平台的.....2015/09/19

第二十四章 核心编译

我们说的 Linux 其实指的就是核心 （kernel） 而已。这个核心控制你主机的所有硬件并提供系统所有的功能， 所以说，他重不重要啊！我们开机的时候其实就是利用开机管理程序载入这个核心文件来侦测硬件， 在核心载入适当的驱动程序后.....2015/10/20

第零章、计算机概论

最近更新日期：20//

由过去的经验当中，鸟哥发现到因为兴趣或生活所逼而必须要接触 Linux 的朋友，很多可能并非信息相关专业出身， 因此对于电脑软/硬件方面的概念不熟。然而操作系统这种咚咚跟硬件有相当程度的关连性， 所以，如果不了解一下计算机概论，要很快的了解 Linux 的概念是有点难度的。因此，鸟哥就自作聪明的新增一个小章节来谈谈计概啰！ 因为鸟哥也不是信息相关学科出身，所以，写的不好的地方请大家多多指教啊！^_^

0.1 电脑：辅助人脑的好工具

现在的人们几乎无时无刻都会碰电脑！不管是桌面电脑（台式机）、笔记本电脑（笔记本）、平板电脑、智能手机等等，这些东西都算是电脑。 虽然接触的这么多，但是，你了解电脑里面的元件有什么吗？以台式机来说，电脑的机箱里面含有什么元件？不同的电脑可以应用在哪些工作？ 你生活周遭有哪些电器用品内部是含有电脑相关元件的？下面我们就来谈一谈这些东西呢！

所谓的电脑就是一种计算机，而计算机其实是：“接受使用者输入指令与数据，经由中央处理器的数学与逻辑单元运算处理后， 以产生或储存成有用的信息”。因此，只要有输入设备 （不管是键盘还是触摸屏） 及输出设备 （例如电脑屏幕或直接由打印机打印出来），让你可以输入数据使该机器产生信息的， 那就是一部计算机了。



Tips 电脑可以协助人们进行大量的运算！以前如果要计算化学反应式都得要算个老半天，有了电脑仿真软件后，就有不一样的情况发生了！ 以下图为例，鸟哥的工作中，有一项是需要将人们排放的空气污染物带入电脑模式进行仿真后，计算出可能产生的空气污染并得到空气品质状态，最后经过分析软件得到各式各样的图表。 经过这些图表的解析，就可以让人们知道什么样的污染排放来源可能会产生什么样的空气品质变化啰。

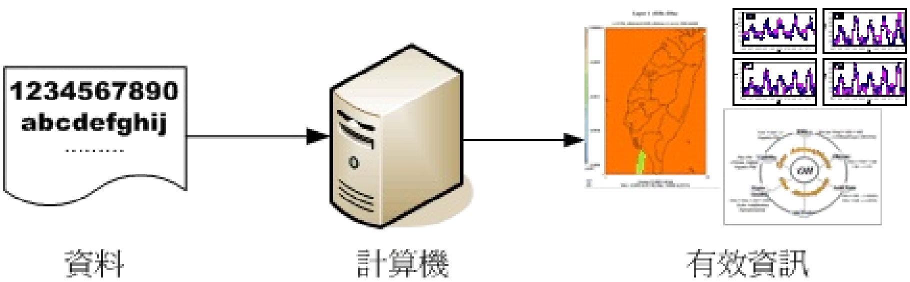


图0.1.1、计算器的功能

好了，根据这个定义你知道哪些东西是计算机了吗？其实包括一般商店用的简易型加减乘除计算器、打电话用的手机、开车用的卫星定位系统 （GPS）、提款用的提款机 （ATM）、你上课会使用的桌上型个人电脑、外出可能会带的笔记本电脑 （包括 notebook 与 netbook），还有近几年 （2015前后） 非常热门的平板电脑与智能手机， 甚至是未来可能会大流行的单版电脑 （Xapple pi, banana pi, Raspberry pi, [1]） 与智能手表，甚至于更多的智能穿戴式电脑[2]等等，这些都是计算机喔！

那么计算机主要的组成元件是什么呢？下面我们以常见的个人电脑主机或服务器工作站主机来作为说明好了。

0.1.1 计算机硬件的五大单元

关于电脑的硬件组成部分，其实你可以观察你的台式机来分析一下，依外观来说这家伙主要可分为三部分，分别是：

- 输入单元：包括键盘、鼠标、读卡机、扫描仪、手写板、触摸屏等等一堆；
- 主机部分：这个就是系统单元，被主机机箱保护住了，里面含有一堆板子、CPU 与内存等；
- 输出单元：例如屏幕、打印机等等

我们主要通过输入设备如鼠标与键盘来将一些数据输入到主机里面，然后再由主机的功能处理成为图表或文章等信息后，将结果传输到输出设备，如屏幕或打印机上面。那主机里面含有什么元件呢？如果你曾经拆开过电脑主机机箱（包括拆开你的智能手机也一样喔！），会发现其实主机里面最重要的就是一块主板，上面安插了中央处理器（CPU）以及内存、硬盘（或记忆卡）还有一些适配卡设备而已。当然大部分智能手机是将这些元件直接焊接在主板上面而不是插卡啦！

整部主机的重点在于中央处理器（Central Processing Unit, CPU），CPU 为一个具有特定功能的芯片，里头含有微指令集，如果你想要让主机进行什么特异的功能，就得要参考这颗 CPU 是否有相关内置的微指令集才可以。由于 CPU 的工作主要在于管理与运算，因此在 CPU 内又可分为两个主要的单元，分别是：算数逻辑单元与控制单元。[3] 其中算数逻辑单元主要负责程序运算与逻辑判断，控制单元则主要在协调各周边元件与各单元间的工作。

既然 CPU 的重点是在进行运算与判断，那么要被运算与判断的数据是从哪里来的？CPU 读取的数据都是从内存来的！内存内的数据则是从输入单元所传输进来！而 CPU 处理完毕的数据也必须要先写回内存中，最后数据才从内存传输到输出单元。



Tips 为什么我们都会说，要加快系统性能，通常将内存容量加大就可以获得相当好的成效？如同下图以及上面的说明，因为所有的数据都要经过内存的传输，所以内存的容量如果太小，数据高速缓存就不足～影响性能相当大啊！尤其针对 Linux 作为服务器的环境下！这点要特别记忆喔！

综合上面所说的，我们会知道其实电脑是由几个单元所组成的，包括输入单元、输出单元、CPU 内部的控制单元、算数逻辑单元与内存五大部分。这几个东西的相关性如下所示：

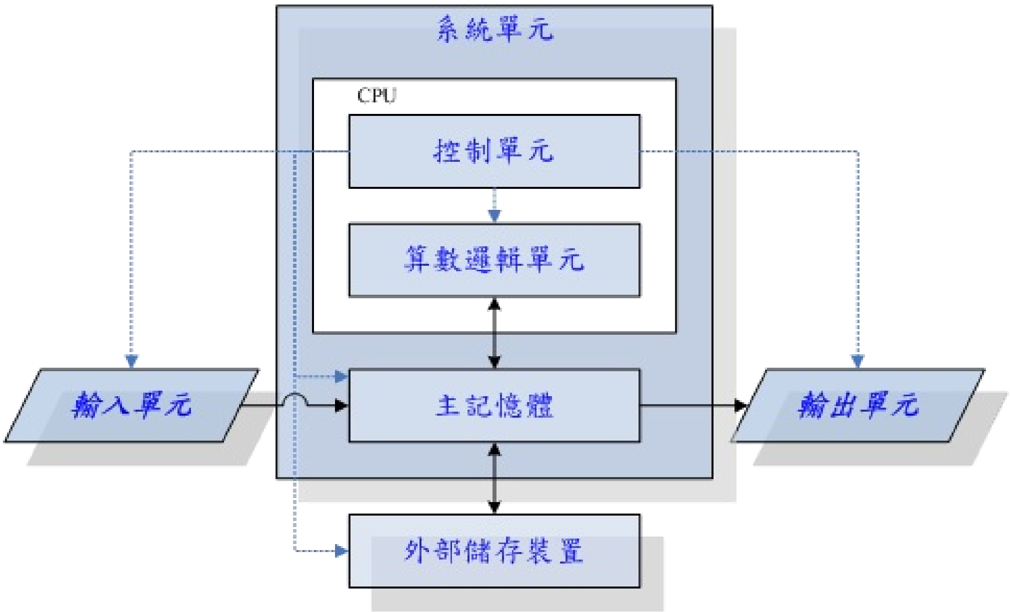


图0.1.2、电脑的五大单元[4]

上面图示中的“系统单元”其实指的就是电脑机箱内的主要元件，而重点在于 CPU 与内存。 特别要看的是实线部分的传输方向，基本上数据都是流经过内存再转出去的！ 至于数据会流进/流出内存则是 CPU 所发布的控制命令！而 CPU 实际要处理的数据则完全来自于内存 （不管是程序还是一般文件数据）！这是个很重要的概念喔！ 这也是为什么当你的内存不足时，系统的性能就很糟糕！也是为什么现在人们买智能手机时，对于可用内存的要求都很高的原因！

而由上面的图示我们也能知道，所有的单元都是由 CPU 内部的控制单元来负责协调的，因此 CPU 是整个电脑系统的最重要部分！ 那么目前世界上有哪些主流的 CPU 呢？是否刚刚我们谈到的硬件内全部都是相同的 CPU 架构呢？下面我们就来谈一谈。

0.1.2 一切设计的起点： CPU 的架构

如前面说过的，CPU 其实内部已经含有一些微指令，我们所使用的软件都要经过 CPU 内部的微指令集来达成才行。 那这些指令集的设计主要又被分为两种设计理念，这就是目前世界上常见到的两种主要 CPU 架构， 分别是：精简指令集 （RISC） 与复杂指令集 （CISC） 系统。下面我们就来谈谈这两种不同 CPU 架构的差异啰！

- 精简指令集 （Reduced Instruction Set Computer, RISC） ：[5]

这种 CPU 的设计中，微指令集较为精简，每个指令的执行时间都很短，完成的动作也很单纯，指令的执行性能较佳； 但是若要做复杂的事情，就要由多个指令来完成。常见的 RISC 微指令集 CPU 主要例如甲骨文 （Oracle） 公司的 SPARC 系列、 IBM 公司的 Power Architecture （包括 PowerPC） 系列、与安谋公司 （ARM Holdings） 的 ARM CPU 系列等。

在应用方面，SPARC CPU 的电脑常用于学术领域的大型工作站中，包括银行金融体系的主要服务器也都有这类的电脑架构； 至于PowerPC架构的应用上，例如索尼（Sony）公司出产的Play Station 3（PS3）就是使用PowerPC架构的Cell处理器； 那安谋的 ARM 呢？你常使用的各厂牌手机、PDA、导航系统、网络设备（交换器、路由器等）等，几乎都是使用 ARM 架构的 CPU 喔！ 老实说，目前世界上使用范围最广的 CPU 可能就是 ARM 这种架构的呢！[6]

- 复杂指令集（Complex Instruction Set Computer, CISC）：[7]

与RISC不同的，CISC在微指令集的每个小指令可以执行一些较低阶的硬件操作，指令数目多而且复杂， 每条指令的长度并不相同。因为指令执行较为复杂所以每条指令花费的时间较长， 但每条个别指令可以处理的工作较为丰富。常见的CISC微指令集CPU主要有AMD、Intel、VIA等的x86架构的CPU。

由于AMD、Intel、VIA所开发出来的x86架构CPU被大量使用于个人电脑（Personal computer）用途上面， 因此，个人电脑常被称为x86架构的电脑！那为何称为x86架构[8]呢？这是因为最早的那颗Intel发展出来的CPU代号称为8086，后来依此架构又开发出80286, 80386...， 因此这种架构的CPU就被称为x86架构了。

在2003年以前由Intel所开发的x86架构CPU由8位升级到16、32位，后来AMD依此架构修改新一代的CPU为64位， 为了区别两者的差异，因此64位的个人电脑CPU又被统称为x86_64的架构喔！



Tips 所谓的位指的是CPU一次数据读取的最大量！64位CPU代表CPU一次可以读写64bits这么多的数据，32位CPU则是CPU一次只能读取32位的意思。 因为CPU读取数据量有限制，因此能够从内存中读写的数据也就有所限制。所以，一般32位的CPU所能读写的最大数据量，大概就是4GB左右。

那么不同的x86架构的CPU有什么差异呢？除了CPU的整体结构（如第二层高速缓存、每次运行可执行的指令数等）之外， 主要是在于微指令集的不同。新的x86的CPU大多含有很先进的微指令集， 这些微指令集可以加速多媒体程序的运行，也能够加强虚拟化的性能，而且某些微指令集更能够增加能源效率， 让CPU耗电量降低呢！由于电费越来越高，购买电脑时，除了整体的性能之外， 节能省电的CPU特色也可以考虑喔！

例题：最新的Intel/AMD的x86架构中，请查询出多媒体、虚拟化、省电功能各有哪些重要的微指令集？（仅供参考）答：

- 多媒体微指令集：MMX, SSE, SSE2, SSE3, SSE4, AMD-3DNow!
- 虚拟化微指令集：Intel-VT, AMD-SVM

- 省电功能：Intel-SpeedStep, AMD-PowerNow!
- 64/32位相容技术：AMD-AMD64, Intel-EM64T

0.1.3 其他单元的设备

五大单元中最重要的控制、算术逻辑被整合到 CPU 的封装中，但系统当然不可能只有 CPU 啊！那其他三个重要电脑单元的设备还有哪些呢？ 其实在主机机箱内的设备大多是通过主板（main board） 连接在一块，主板上面有个链接沟通所有设备的芯片组，这个芯片组可以将所有单元的设备链接起来， 好让 CPU 可以对这些设备下达命令。其他单元的重要设备主要有：

- 系统单元：如图 0.1.2 所示，系统单元包括 CPU 与内存及主板相关元件。而主板上头其实还有很多的连接接口与相关的适配卡，包括鸟哥近期常使用的 PCI-E 10G 网卡、 磁盘阵列卡、还有显卡等等。尤其是显卡，这东西对于玩3D游戏来说是非常重要的一环，他与显示的精致度、色彩与分辨率都有关系。

- 存储单元：包括内存 （main memory, RAM） 与辅助内存，其中辅助内存其实就是大家常听到的“储存设备”啰！包括硬盘、软盘、光盘、磁带等等的。

- 输入、输出单元：同时涵盖输入输出的设备最常见的大概就是触摸屏了。至于单纯的输入设备包括前面提到的键盘鼠标之外，目前的体感设备也是重要的输入设备喔！ 至于输出设备方面，除了屏幕外，打印机、音效喇叭、HDMI电视、投影机、蓝牙耳机等等，都算喔！

更详细的各项主机与周边设备我们将在下个小节进行介绍！在这里我们先来了解一下各元件的关系啰！那就是，电脑是如何运行的呢？

0.1.4 运行流程

如果不是很了解电脑的运行流程的话，鸟哥拿个简单的想法来思考好了～假设电脑是一个人体，那么每个元件对应到那个地方呢？可以这样思考：

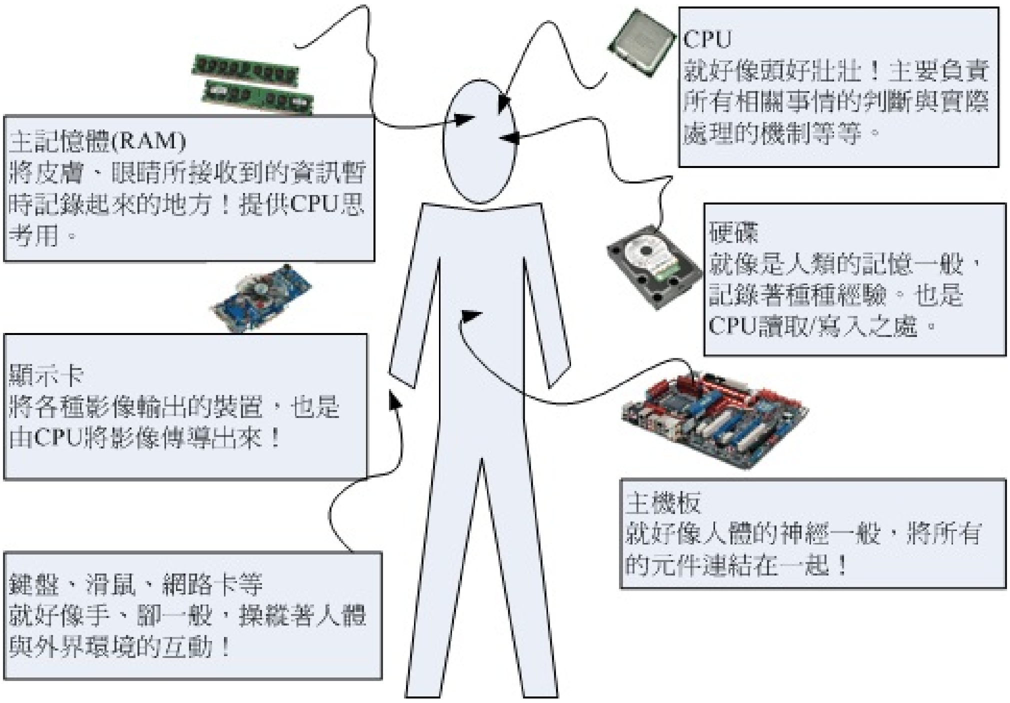


图0.1.3、各元件运行

- CPU=脑袋瓜子：每个人会作的事情都不一样（微指令集的差异），但主要都是通过脑袋瓜子来进行判断与控制身体各部分的活动；
- 内存=脑袋中放置正在被思考的数据的区块：在实际活动过程中，我们的脑袋瓜子需要有外界刺激的数据 （例如光线、环境、语言等） 来分析，那这些互动数据暂时存放的地方就是内存，主要是用来提供给脑袋瓜子判断用的信息。
- 硬盘=脑袋中放置回忆的记忆区块：跟刚刚的内存不同，内存是提供脑袋目前要思考与处理的信息，但是有些生活琐事或其他没有要立刻处理的事情， 就当成回忆先放置到脑袋的记忆深处吧！那就是硬盘！主要目的是将重要的数据记录起来，以便未来将这些重要的经验再次的使用；
- 主板=神经系统：好像人类的神经一样，将所有重要的元件连接起来，包括手脚的活动都是脑袋瓜子发布命令后， 通过神经（主板）传导给手脚来进行活动啊！
- 各项周边设备=人体与外界沟通的手、脚、皮肤、眼睛等：就好像手脚一般，是人体与外界互动的重要关键！
- 显卡=脑袋中的影像：将来自眼睛的刺激转成影像后在脑袋中呈现，所以显卡所产生的数据来源也是CPU控制的。
- 电源供应器 （Power）=心脏：所有的元件要能运行得要有足够的电力供给才行！这电力供给就好像心脏一样，如果心脏不够力， 那么全身也就无法动弹的！心脏不稳定呢？那你的身体当然可能断断续续的～不稳定！

由这样的关系图当中，我们知道整个活动中最重要的就是脑袋瓜子！ 而脑袋瓜子当中与现在正在进行的工作有关的就是CPU与内存！任何外界的接触都必须要由脑袋瓜子中的内存记录下来， 然后给脑袋中的CPU依据这些数据进行判断后，再发布命令给各个周边设备！如果需要用到过去的经验， 就得由过去的经验（硬盘）当中读取啰！

也就是说，整个人体最重要的地方就是脑袋瓜子，同样的，整部主机当中最重要的就是CPU与内存， 而CPU的数据来源通通来自于内存，如果要由过去的经验来判断事情时， 也要将经验（硬盘）挪到目前的记忆（内存）当中，再交由CPU来判断喔！这点得要再次的强调啊！ 下个章节当中，我们就对目前常见的个人电脑各个元件来进行说明啰！

0.1.5 电脑按用途分类

知道了电脑的基本组成与周边设备，也知道其实电脑的CPU种类非常的多，再来我们想要了解的是，电脑如何分类？ 电脑的分类非常多种，如果以电脑的复杂度与运算能力进行分类的话，主要可以分为这几类：

- 超级计算机（Supercomputer） 超级计算机是运行速度最快的电脑，但是他的维护、操作费用也最高！主要是用于需要有高速计算的计划中。 例如：国防军事、气象预测、太空科技，用在仿真的领域较多。详情也可以参考： 国家高速网络与计算中心 http://www.nchc.org.tw的介绍！ 至于全世界最快速的前500大超级计算机，则请参考：http://www.top500.org。

- 大型计算机（Mainframe Computer） 大型计算机通常也具有数个高速的CPU，功能上虽不及超级计算机，但也可用来处理大量数据与复杂的运算。 例如大型企业的主机、全国性的证券交易所等每天需要处理数百万笔数据的企业机构， 或者是大型企业的数据库服务器等等。

- 迷你电脑（Minicomputer） 迷你电脑仍保有大型计算机同时支持多使用者的特性，但是主机可以放在一般作业场所， 不必像前两个大型计算机需要特殊的空调场所。通常用来作为科学研究、工程分析与工厂的流程管理等。

- 工作站（Workstation） 工作站的价格又比迷你电脑便宜许多，是针对特殊用途而设计的电脑。在个人电脑的性能还没有提升到目前的状况之前， 工作站电脑的性能/价格比是所有电脑当中较佳的，因此在学术研究与工程分析方面相当常见。

- 微电脑（Microcomputer） 个人电脑就属于这部份的电脑分类，也是我们本章主要探讨的目标！体积最小，价格最低，但功能还是五脏俱全的！ 大致又可分为桌上型、笔记型等等。

若光以性能来说，目前的个人电脑性能已经够快了，甚至已经比工作站等级以上的电脑运算速度还要快！ 但是工作站电脑强调的是稳定不死机，并且运算过程要完全正确，因此工作站以上等级的电脑在设计时的考虑与个人电脑并不相同啦！ 这也是为啥工作站等级以上的电脑售价较贵的原因。

0.1.6 电脑上面常用的计算单位 （容量、速度等）

电脑的运算能力除了 CPU 微指令集设计的优劣之外，但主要还是由速度来决定的。至于存放在电脑储存设备当中的数据容量也是有单位的。

- 容量单位

电脑对数据的判断主要依据有没有通电来记录信息，所以理论上对于每一个纪录单位而言，它只认识 0 与 1 而已。0/1 这个二进制的的单位我们称为 bit。 但 bit 实在太小了，所以在储存数据时每份简单的数据都会使用到 8 个 bits 的大小来记录，因此定义出 Byte 这个单位，他们的关系为：

> 1 Byte = 8 bits

不过同样的，Byte 还是太小了，在较大的容量情况下，使用 Byte 相当不容易判断数据的大小，举例来说，1000000 Bytes 这样的显示方式你能够看得出有几个零吗？所以后来就有一些常见的简化单位表达式，例如 K 代表 1024Byte，M 代表 1024K 等。 而这些单位在不同的进位制下有不同的数值表示，下面就列出常见的单位与进位制对应：

进位制	Kilo	Mega	Giga	Tera	Peta	Exa	Zetta
二进制	1024	1024K	1024M	1024G	1024T	1024P	1024E
十进制	1000	1000K	1000M	1000G	1000T	1000P	1000E

一般来说，文件大小使用的是二进制的方式，所以 1GBytes 的文件大小实际上为：1024x1024x1024Bytes 这么大！ 速度单位则常使用十进制，例如 1GHz 就是 1000x1000x1000Hz 的意思。



Tips 那么什么是“进位”呢？以人类最常用的十进制为例，每个“位置”上面最多仅能有一个数值，这个数值不可以比 9 还要大！那比 9 还大怎办？ 就用“第二个位置来装一个新的 1 ”！所以， 9 还是只有一个位置， 10 则是用了两个位置了。好了那如果是 16 进位怎办？由于每个位置只能出现一个数值， 但是数字仅有 0~9 而已啊！因此 16 进位中，就以 A 代表 10 的意思，以 B 代表 11 的意思，所以 16 进位就是 0~9, a, b, c, d, e, f，有没有看到， “每个位置最多还是只有一个数值而已”喔！好了，那回来谈谈二进制。因为每个位置只能有 0, 1 而已，不能出现 2 （逢 2 进一位） 啦！这样了解乎？

- 速度单位

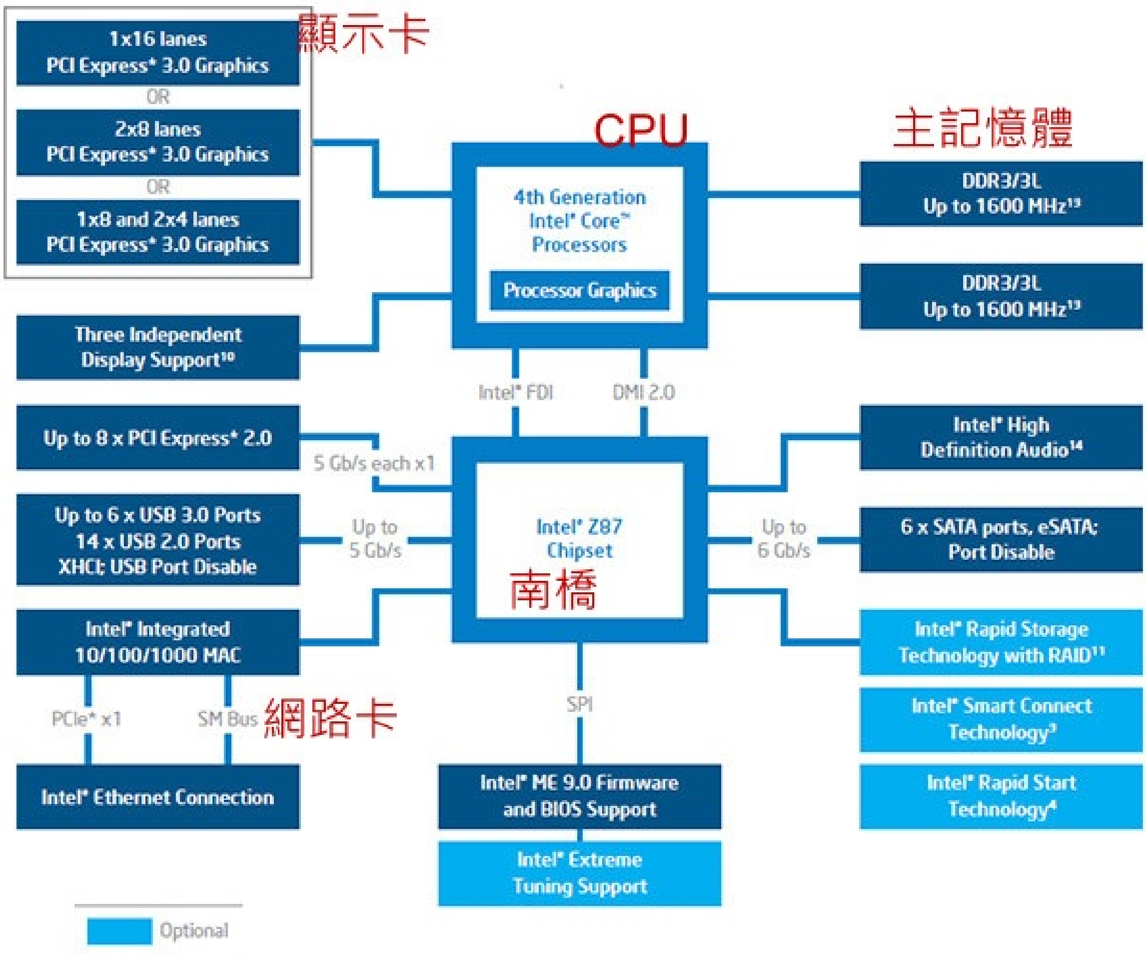
CPU的运算速度常使用 MHz 或者是 GHz 之类的单位，这个 Hz 其实就是秒分之一。而在网络传输方面，由于网络使用的是 bit 为单位，因此网络常使用的单位为 Mbps 是 Mbits per second，亦即是每秒多少 Mbit。举例来说，大家常听到的 20M/5M 光世代传输速度，如果转成文件大小的 Byte 时，其实理论最大传输值为：每秒 2.5MByte/ 每秒625KByte的下载/上传速度喔！

例题：假设你今天购买了500GB的硬盘一颗，但是格式化完毕后却只剩下460GB左右的容量，这是什么原因？答：因为一般硬盘制造商会使用十进制的单位，所以500GByte代表为500*1000*1000*1000Byte之意。 转成文件的容量单位时使用二进制（1024为底），所以就成为466GB左右的容量了。

硬盘厂商并非要骗人，只是因为硬盘的最小物理量为512Bytes，最小的组成单位为扇区（sector）， 通常硬盘容量的计算采用“多少个sector”，所以才会使用十进制来处理的。相关的硬盘信息在这一章后面会提到的！

0.2 个人电脑架构与相关设备元件

一般消费者常说的电脑通常指的就是x86的个人电脑架构，因此我们有必要来了解一下这个架构的各个元件。事实上，Linux最早在发展的时候，就是依据个人电脑的架构来发展的， 所以真的得要了解一下呢！另外，早期两大主流x86开发商（Intel, AMD）的CPU架构与设计理念都有些许差异。不过互相学习对方长处的结果，就是两者间的架构已经比较类似了。 由于目前市场占有率还是以 Intel 为大宗，因此下面以目前（2015）相对较新的 Intel 主板架构来谈谈：



图

0.2.1、Intel芯片架构

由于主板是链接各元件的一个重要项目，因此在主板上面沟通各部元件的芯片组设计优劣，就会影响性能不少喔！早期的芯片组通常分为两个桥接器来控制各元件的沟通， 分别是：（1）北桥：负责链接速度较快的CPU、内存与显卡接口等元件；（2）南桥：负责连接速度较慢的设备接口， 包括硬盘、USB、网卡等等。（芯片组的南北桥与三国的大小乔没有关系@_@）。不过由于北桥最重要的就是 CPU 与内存之间的桥接，因此目前的主流架构中， 大多将北桥内存控制器整合到 CPU 封装当中了。所以上图你只会看到 CPU 而没有看到以往的北桥芯片喔！



Tips 早期芯片组分南北桥，北桥可以连接 CPU、内存与显卡。只是 CPU 要读写到内存的动作，还需要北桥的支持，也就是 CPU 与内存的交流， 会瓜分掉北桥的总可用带宽，真浪费！因此目前将内存控制器整合到 CPU 后，CPU与内存之间的沟通是直接交流，速度较快之外，也不会消耗更多的带宽！

毕竟目前世界上x86的CPU主要供应商为Intel，所以下面鸟哥将以Intel的主板架构说明各元件啰！ 我们以华硕公司出的主板，型号：Asus Z97-AR 作为一个说明的范例，搭配着主板芯片组逻辑图 0.2.1 的说明，主板各元件如下所示：

图 0.2.2、ASUS 主板 （图片为华硕公司所有）

上述的图片中，主板上面设计的插槽主要有 CPU （Intel LGA 1150 Socket）、内存 （DDR3 3200 support）、显卡接口 （PCIe3.0）、SATA 磁盘插槽 （SATA express）等等。 下面的元件在解说的时候，请参考上述两张图示来印证喔！

0.2.1 执行脑袋运算与判断的 CPU

如同华硕主板示意图上半部的中央部分，那就是CPU插槽。 由于CPU负责大量运算，因此CPU通常是具有相当高发热量的元件。所以如果你曾经拆开过主板， 应该就会看到CPU上头通常会安插一颗风扇来主动散热的。

x86个人电脑的CPU主要供应商为Intel与AMD，目前（2015）主流的CPU都是双核以上的架构了！ 原本的单核心CPU仅有一个运算单元，所谓的多核心则是在一颗CPU封装当中嵌入了两个以上的运算核心， 简单的说，就是一个实体的CPU外壳中，含有两个以上的CPU单元就是了。

不同的CPU型号大多具有不同的脚位（CPU上面的插脚），能够搭配的主板芯片组也不同，所以当你想要将你的主机升级时，不能只考虑CPU，你还得要留意你的主板上面所支持的CPU型号喔！ 不然买了最新的CPU也不能够安插在你的旧主板上头的！目前主流的CPU有Intel的 i3/i5/i7 系列产品中，甚至先后期出厂的类似型号的脚位也不同， 例如 i7-2600 使用 LGA1155 脚位而 i7-4790 则使用 FCLGA1150 脚位，挑选时必须要很小心喔！

我们前面谈到CPU内部含有微指令集，不同的微指令集会导致CPU工作效率的优劣。除了这点之外， CPU性能的比较还有什么呢？那就是CPU的频率了！什么是频率呢？简单的说， 频率就是CPU每秒钟可以进行的工作次数。 所以频率越高表示这颗CPU单位时间内可以作更多的事情。举例来说，Intel的 i7-4790 CPU频率为3.6GHz， 表示这颗CPU在一秒内可以进行3.6x109次工作，每次工作都可以进行少数的指令运行之意。



Tips 注意，不同的CPU之间不能单纯的以频率来判断运算性能喔！这是因为每颗CPU的微指令集不相同，架构也不见得一样，可使用的第二层高速缓存及其计算机制可能也不同， 加上每次频率能够进行的工作指令数也不同！所以，频率目前仅能用来比较同款CPU的速度！

- CPU的工作频率：外频与倍频

早期的 CPU 架构主要通过北桥来链接系统最重要的 CPU、内存与显卡设备。因为所有的设备都得通过北桥来链接，因此每个设备的工作频率应该要相同。 于是就有所谓的前端总线（FSB） 这个东西的产生。但因为 CPU 的运算速度比其他的设备都要来的快，又为了要满足 FSB 的频率，因此厂商就在 CPU 内部再进行加速， 于是就有所谓的外频与倍频了。

总结来说，在早期的 CPU 设计中，所谓的外频指的是CPU与外部元件进行数据传输时的速度，倍频则是 CPU 内部用来加速工作性能的一个倍数， 两者相乘才是CPU的频率速度。例如 Intel Core 2 E8400 的内频为 3.0GHz，而外频是333MHz，因此倍频就是9倍啰！（3.0G=333Mx9, 其中1G=1000M）



Tips 很多计算机硬件玩家很喜欢玩“超频”，所谓的超频指的是： 将CPU的倍频或者是外频通过主板的设置功能更改成较高频率的一种方式。但因为CPU的倍频通常在出厂时已经被锁定而无法修改， 因此较常被超频的为外频。 举例来说，像上述3.0GHz的CPU如果想要超频，可以将他的外频333MHz调整成为400MHz，但如此一来整个主板的各个元件的运行频率可能都会被增加成原本的1.333倍（4/3）， 虽然CPU可能可以到达3.6GHz，但却因为频率并非正常速度，故可能会造成死机等问题。

但如此一来所有的数据都被北桥卡死了，北桥又不可能比 CPU 更快，因此这家伙常常是系统性能的瓶颈。为了解决这个问题，新的 CPU 设计中， 已经将内存控制器整合到 CPU 内部，而链接 CPU 与内存、显卡的控制器的设计，在Intel部份使用 QPI （Quick Path Interconnect） 与 DMI 技术，而 AMD 部份则使用 Hyper Transport 了，这些技术都可以让 CPU 直接与内存、显卡等设备分别进行沟通，而不需要通过外部的链接芯片了。

因为现在没有所谓的北桥了 （整合到 CPU 内） ，因此，CPU 的频率设计就无须考虑得要同步的外频，只需要考虑整体的频率即可。 所以，如果你经常有查阅自己 CPU 频率的习惯，当使用 cpu-z [9] 这个软件时，应该会很惊讶的发现到，怎么外频变成 100MHz 而倍频可以到达 30 以上！相当有趣呢！



Tips 现在 Intel 的 CPU 会主动帮你超频喔！例如 i7-4790 这颗 CPU 的规格 [10] 中，基本频率为 3.6GHz，但是最高可自动超频到 4GHz 喔！ 通过的是 Intel 的 turbo 技术。同时，如果你没有大量的运算需求，该 CPU 频率会降到 1.xGHz 而已，借此达到节能省电的目的！所以，各位好朋友， 不需要自己手动超频了！Intel 已经自动帮你进行超频了...所以，如果你用 cpu-z 观察 CPU 频率，发现该频率会一直自动变动，很正常！你的系统没坏掉！

- 32位与64位的CPU与总线“宽度”

从前面的简易说明中，我们知道 CPU 的各项数据通通得要来自于内存。因此，如果内存能提供给 CPU 的数据量越大的话，当然整体系统的性能应该也会比较快！ 那如何知道内存能提供的数据量呢？此时还是得要借由 CPU 内的内存控制芯片与内存间的传输速度“前端总线速度（Front Side Bus, FSB） 来说明。

与 CPU 的频率类似的，内存也是有其工作的频率，这个频率限制还是来自于 CPU 内的内存控制器所决定的。以图0.2.1 为例， CPU 内置的内存控制芯片对内存的工作频率最高可达到 1600MHz。这只是工作频率（每秒几次）。一般来说，每次频率能够传输的数据量，大多为 64 位，这个 64 位就是所谓的“宽度”了！ 因此，在图0.2.1 这个系统中，CPU可以从内存中取得的最快带宽就是 1600MHz *64bit = 1600MHz* 8 Bytes = 12.8GByte/s。

与总线宽度相似的，CPU每次能够处理的数据量称为字组大小（word size）， 字组大小依据CPU的设计而有32位与64位。我们现在所称的电脑是32或64位主要是依据这个 CPU解析的字组大小而来的！早期的32位CPU中，因为CPU每次能够解析的数据量有限， 因此由内存传来的数据量就有所限制了。这也导致32位的CPU最多只能支持最大到4GBytes的内存。



Tips 得利于北桥整合到 CPU 内部的设计，CPU 得以“个别”跟各个元件进行沟通！因此，每种元件与 CPU 的沟通具有很多不同的方式！例如内存使用系统总线带宽来与 CPU 沟通。而显卡则通过PCI-E的序列信道设计来与CPU沟通喔！详细说明我们在本章稍后的主板部份再来谈谈。

- CPU等级

由于x86架构的CPU在Intel的Pentium系列（1993年）后就有不统一的脚位与设计，为了将不同种类的CPU规范等级， 所以就有i386,i586,i686等名词出现了。基本上，在Intel Pentium MMX与AMD K6年代的CPU称为i586等级， 而Intel Celeron与AMD Athlon（K7）年代之后的

32位CPU就称为i686等级。 至于目前的64位CPU则统称为x86_64等级。

目前很多的程序都有对CPU做最优化的设计，万一哪天你发现一些程序是注明给x86_64的CPU使用时， 就不要将他安装在686以下等级的电脑中，否则可是会无法执行该软件的！ 不过，在x86_64的硬件下倒是可以安装386的软件喔！也就是说，这些东西具有向下相容的能力啦！

- 超线程 （Hyper-Threading, HT）

我们知道现在的 CPU 至少都是两个核心以上的多核心 CPU 了，但是 Intel 还有个很怪的东西，叫做 CPU 的超线程 （Hyper-Threading） 功能！ 那个是啥鬼东西？我们知道现在的 CPU 运算速度都太快了，因此运算核心经常处于闲置状态下。而我们也知道现在的系统大多都是多任务的系统， 同时间有很多的程序会让 CPU 来执行。因此，若 CPU 可以假象的同时执行两个程序，不就可以让系统性能增加了吗？反正 CPU 的运算能力还是没有用完啊！

那是怎么达成的啊这个 HT 功能？强者鸟哥的同事蔡董大大用个简单的说明来解释。在每一个 CPU 内部将重要的寄存器 （register） 分成两群， 而让程序分别使用这两群寄存器。也就是说，可以有两个程序“同时竞争 CPU 的运算单元”，而非通过操作系统的多任务切换！ 这一过程就会让 CPU 好像“同时有两个核心”的模样！因此，虽然大部分 i7 等级的 CPU 其实只有四个实体核心，但通过 HT 的机制， 则操作系统可以抓到八个核心！并且让每个核心逻辑上分离，就可以同时运行八个程序了。

虽然很多研究与测试中，大多发现 HT 虽然可以提升性能，不过，有些情况下却可能导致性能降低喔！因为，实际上明明就仅有一个运算单元嘛！ 不过在鸟哥使用数值模式的情况下，因为鸟哥操作的数值模式主要为平行运算功能，且运算通常无法达到 100% 的 CPU 使用率，通常仅有大约60%运算量而已。 因此在鸟哥的实作过程中，这个 HT 确实提升相当多的性能！至少应该可以节省鸟哥大约30%~50%的等待时间喔！不过网络上大家的研究中， 大多说这个是 case by case，而且使用的软件影响很大！所以，在鸟哥的例子是启用 HT 帮助很大！您的案例就得要自行研究啰！

0.2.2 内存

如同图0.2.2、华硕主板示意图中的右上方部分的那四根插槽，那就是内存的插槽了。 内存插槽中间通常有个突起物将整个插槽稍微切分成为两个不等长的距离， 这样的设计可以让使用者在安装内存时，不至于前后脚位安插错误，是一种防呆的设计喔。

前面提到CPU所使用的数据都是来自于内存（main memory），不论是软件程序还是数据，都必须要读入内存后CPU才能利用。 个人电脑的内存主要元件为动态随机存取内存（Dynamic Random Access Memory, DRAM）， 随机存取内存只有在通电时才能记录与使用，断电后数据就消失了。因此我们也称这种RAM为挥发性内存。

DRAM根据技术的更新又分好几代，而使用上较广泛的有所谓的SDRAM与DDR SDRAM两种。 这两种内存的差别除了在于脚位与工作电压上的不同之外，DDR是所谓的双倍数据传送速度（Double Data Rate）， 他可以在一次工作周期中进行两次数据的传送，感觉上就好像

是CPU的倍频啦！ 所以传输频率方面比SDRAM还要好。新一代的PC大多使用DDR内存了。下表列出SDRAM与DDR SDRAM的型号与频率及带宽之间的关系。[11]

SDRAM/DDR	型号	数据宽度（bit）	内部频率（MHz）	频率速度	带宽（频率x宽度）
SDRAM	PC100	64	100	100	800MBytes/sec
SDRAM	PC133	64	133	133	1064MBytes/sec
DDR	DDR-266	64	133	266	2.1GBytes/sec
DDR	DDR-400	64	200	400	3.2GBytes/sec
DDR	DDR2-800	64	200	800	6.4GBytes/sec
DDR	DDR3-1600	64	200	1600	12.8GBytes/sec

DDR SDRAM又依据技术的发展，有DDR, DDR2, DDR3, DDR4等等，其中，DDR2 的频率倍数则是 4 倍而DDR3 则是 8 倍喔！ 目前鸟哥用到服务器等级的内存，已经到 DDR4 了耶！超快超快！



Tips 在图 0.2.1 中，内存的规格内提到 DDR3/DDR3L 同时支持，我们知道 DDR3 了，那 DDR3L 是啥鬼？ 为了节省更多的电力，新的制程中降低了内存的操作电压，因此 DDR3 标准电压为 1.5V，但 DDR3L 则仅须 1.35V 喔！通常可以用在耗电量需求更低的笔记本中！ 但并非所有的系统都同步支持！这就得要看主板的支持规格啰！否则你买了 DDR3L 安插在不支持的主板上，DDR3L 内存是可能会烧毁的喔！

内存除了频率/带宽与型号需要考虑之外，内存的容量也是很重要的喔！ 因为所有的数据都得要载入内存当中才能够被CPU判读，如果内存容量不够大的话将会导致某些大容量数据无法被完整的载入， 此时已存在内存当中但暂时没有被使用到的数据必须要先被释放，使得可用内存容量大于该数据，那份新数据才能够被载入呢！ 所以，通常越大的内存代表越快速的系统，这是因为系统不用常常释放一些内存内部的数据。 以服务器来说，内存的容量有时比CPU的速度还要来的重要的！

- 多通道设计

由于所有的数据都必须要存放在内存，所以内存的数据宽度当然是越大越好。 但传统的总线宽度一般大约仅达64位，为了要加大这个宽度，因此芯片组厂商就将两个内存汇整在一起，如果一支内存可达64位，两支内存就可以达到128位了，这就是双通道的设计理念。

如上所述，要启用双通道的功能你必须要安插两支（或四支）内存，这两支内存最好连型号都一模一样比较好， 这是因为启动双通道内存功能时，数据是同步写入/读出这一对内存中，如此才能够提升整体的带宽啊！ 所以当然除了容量大小要一致之外，型号也最好相同啦！

你有没有发现图 0.2.2、华硕主板示意图上那四根内存插槽的颜色呢？是否分为两种颜色，且两两成对？ 为什么要这样设计？答出来了吗？是啦！这种颜色的设计就是为了双通道来的！要启动双通道的功能时， 你必须要将两根容量相同的内存插在相同颜色的插槽当中喔！



Tips 服务器所需要的速度更快！因此，除了双通道之外，中阶服务器也经常提供三信道，甚至四信道的内存环境！ 例如 2014 年推出的服务器用 E5-2650 v3 的 Intel CPU 中，它可以接受的最大信道数就是四信道且为 DDR4 喔！

- DRAM与SRAM

除了内存条之外，事实上整部个人电脑当中还有许许多多的内存存在喔！最为我们所知的就是CPU内的第二层高速缓存内存。 我们现在知道CPU的数据都是由内存提供，但CPU到内存之间还是得要通过内存控制器啊！ 如果某些很常用的程序或数据可以放置到CPU内部的话，那么CPU数据的读取就不需要跑到内存重新读取了！ 这对于性能来说不就可以大大的提升了？这就是第二层高速缓存的设计概念。第二层高速缓存与内存及CPU的关系如下图所示：

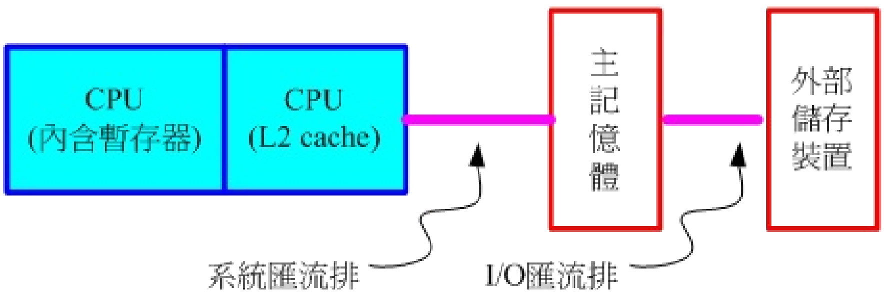


图0.2.3、内存相关性

因为第二层高速缓存（L2 cache）整合到CPU内部，因此这个L2内存的速度必须要CPU频率相同。 使用DRAM是无法达到这个频率速度的，此时就需要静态随机存取内存（Static Random Access Memory, SRAM）的帮忙了。 SRAM在设计上使用的电晶体数量较多，价格较高，且不易做成大容量，不过由于其速度快， 因此整合到CPU内成为高速缓存内存以加快数据的存取是个不错的方式喔！新一代的CPU都有内置容量不等的L2高速缓存在CPU内部，以加快CPU的运行性能。

- 只读存储器（ROM）

主板上面的元件是非常多的，而每个元件的参数又具有可调整性。举例来说，CPU与内存的频率是可调整的； 而主板上面如果有内置的网卡或者是显卡时，该功能是否要启动与该功能的各项参数， 是被记录到主板上头的一个称为CMOS的芯片上，这个芯片需要借着额外的电

源来发挥记录功能， 这也是为什么你的主板上面会有一颗电池的缘故。

那CMOS内的数据如何读取与更新呢？还记得你的电脑在开机的时候可以按下[Del]按键来进入一个名为BIOS的画面吧？ BIOS（Basic Input Output System）是一套程序，这套程序是写死到主板上面的一个内存芯片中， 这个内存芯片在没有通电时也能够将数据记录下来，那就是只读存储器（Read Only Memory, ROM）。 ROM是一种非挥发性的内存。另外，BIOS对于个人电脑来说是非常重要的， 因为他是系统在开机的时候首先会去读取的一个小程序喔！

另外，固件（firmware）[12]很多也是使用ROM来进行软件的写入的。 固件像软件一样也是一个被电脑所执行的程序，然而他是对于硬件内部而言更加重要的部分。例如BIOS就是一个固件， BIOS虽然对于我们日常操作电脑系统没有什么太大的关系，但是他却控制着开机时各项硬件参数的取得！ 所以我们会知道很多的硬件上头都会有ROM来写入固件这个软件。

BIOS 对电脑系统来讲是非常重要的，因为他掌握了系统硬件的详细信息与开机设备的选择等等。但是电脑发展的速度太快了， 因此 BIOS 程序码也可能需要作适度的修改才行，所以你才会在很多主板官网找到 BIOS 的更新程序啊！但是 BIOS 原本使用的是无法改写的 ROM ，因此根本无法修正 BIOS 程序码！为此，现在的 BIOS 通常是写入类似闪存 （flash） 或 EEPROM [13] 中。[14]



Tips 很多硬件上面都会有固件喔！例如鸟哥常用的磁盘阵列卡、 10G 的网卡、交换器设备等等！你可以简单的这么想！固件就是绑在硬件上面的控制软件！

0.2.3 显卡

显卡插槽如同图 0.2.2、华硕主板示意图所示，在中左方有个 PCIe 3.0 的项目， 这张主板中提供了两个显卡插槽喔！

显卡又称为VGA（Video Graphics Array），他对于图形影像的显示扮演相当关键的角色。 一般对于图形影像的显示重点在于分辨率与色彩深度，因为每个图像显示的颜色会占用掉内存， 因此显卡上面会有一个内存的容量，这个显存容量将会影响到你的屏幕分辨率与色彩深度的喔！

除了显存之外，现在由于三度空间游戏（3D game）与一些3D动画的流行，因此显卡的“运算能力”越来越重要。 一些3D的运算早期是交给CPU去运行的，但是CPU并非完全针对这些3D来进行设计的，而且CPU平时已经非常忙碌了呢！ 所以后来显卡厂商直接在显卡上面嵌入一个3D加速的芯片，这就是所谓的GPU称谓的由来。

显卡主要也是通过CPU的控制芯片来与CPU、内存等沟通。如前面提到的，对于图形影像（尤其是3D游戏）来说， 显卡也是需要高速运算的一个元件，所以数据的传输也是越快越好！因此显卡的规格由早期的PCI导向AGP， 近期AGP又被PCI-Express规格所取代了。如前面华硕主板图示当中看到的就是PCI-Express的插槽。 这些插槽最大的差异就是在数据传输的带宽了！如下所示：

规格	宽度	速度	带宽
PCI	32 bits	33 MHz	133 MBytes/s
PCI 2.2	64 bits	66 MHz	533 MBytes/s
PCI-X	64 bits	133 MHz	1064 MBytes/s
AGP 4x	32 bits	66x4 MHz	1066 MBytes/s
AGP 8x	32 bits	66x8 MHz	2133 MBytes/s
PCIe 1.0 x1	无	无	250 MBytes/s
PCIe 1.0 x8	无	无	2 GBytes/s
PCIe 1.0 x16	无	无	4 GBytes/s

比较特殊的是，PCIe（PCI-Express）使用的是类似管线的概念来处理，在 PCIe 第一版（PCIe 1.0） 中，每条管线可以具有250MBytes/s的带宽性能， 管线越多（通常设计到 x16 管线）则总带宽越高！另外，为了提升更多的带宽，因此 PCIe 还有进阶版本，目前主要的版本为第三版，相关的带宽如下：[15]

规格	**1x**带宽	**16x**带宽
PCIe 1.0	250MByte/s	4GByte/s
PCIe 2.0	500MByte/s	8GByte/s
PCIe 3.0	~1GByte/s	~16GByte/s
PCIe 4.0	~2GByte/s	~32GByte/s

若以图0.2.2的主板为例，它使用的是 PCIe 3.0 的 16x，因此最大带宽就可以到达接近 32GBytes/s 的传输量！ 比起AGP是快很多的！好可怕的传输量....

如果你的主机是用来打3D游戏的，那么显卡的选购是非常重要喔！如果你的主机是用来做为网络服务器的， 那么简单的入门级显卡对你的主机来说就非常够用了！因为网络服务器很少用到3D与图形影像功能。

例题：假设你的桌面使用1024x768分辨率，且使用全彩（每个像素占用3Bytes的容量），请问你的显卡至少需要多少内存才能使用这样的彩度？答：因为1024x768分辨率中会有786432个像素，每个像素占用3Bytes，所以总共需要2.25MBytes以上才行！ 但如果考虑屏幕的更新率（每秒钟屏幕的更新次数），显卡的内存还是越大越好！

除了显卡与主板的连接接口需要知道外，那么显卡是通过什么格式与电脑屏幕 （或电视） 连接的呢？目前主要的连接接口有：

- D-Sub （VGA端子）：为较早之前的连接接口，主要为 15 针的连接，为模拟讯号的传输，当初设计是针对传统图像管屏幕而来。 主要的规格有标准的 640x350px @70Hz、1280x1024px @85Hz 及 2048x1536px @85Hz 等。
- DVI：共有四种以上的接头，不过台湾市面上比较常见的为仅提供数码讯号的 DVI-D，以及整合数码与模拟讯号的 DVI-I 两种。DVI 常见于液晶屏幕的链接， 标准规格主要有：1920x1200px @60Hz、 2560x1600px @60Hz 等。
- HDMI：相对于 D-sub 与 DVI 仅能传送影像数据，HDMI 可以同时传送影像与声音，因此被广泛的使用于电视屏幕中！电脑屏幕目前也经常都有支持 HDMI 格式！
- Display port：与 HDMI 相似的，可以同时传输声音与影像。不过这种接口目前在台湾还是比较少屏幕的支持！

0.2.4 硬盘与储存设备

电脑总是需要记录与读取数据的，而这些数据当然不可能每次都由使用者经过键盘来打字！ 所以就需要有储存设备咯。 电脑系统上面的储存设备包括有：硬盘、软盘、MO、CD、DVD、磁带机、U盘（闪存）、还有新一代的蓝光光驱等， 乃至于大型机器的区域网络储存设备（SAN, NAS）等等，都是可以用来储存数据的。而其中最常见的应该就是硬盘了吧！

- 硬盘的物理组成

大家应该都看过硬盘吧！硬盘依据台式机与笔记本电脑而有分为3.5英寸及2.5英寸的大小。我们以3.5英寸的台式机使用硬盘来说明。 在硬盘盒里面其实是由许许多多的圆形盘片、机械手臂、 磁头与主轴马达所组成的，整个内部如同下图所示：



图0.2.4、硬盘物理构造（图片取自维基百科）

实际的数据都是写在具有磁性物质的盘片上头，而读写主要是通过在机械手臂上的磁头（head）来达成。 实际运行时， 主轴马达让盘片转动，然后机械手臂可伸展让磁头在盘片上头进行读写的动作。 另外，由于单一盘片的容量有限，因此有的硬盘内部会有两个以上的盘片喔！

- 盘片上的数据

既然数据都是写入盘片上头，那么盘片上头的数据又是如何写入的呢？其实盘片上头的数据有点像下面的图示所示：

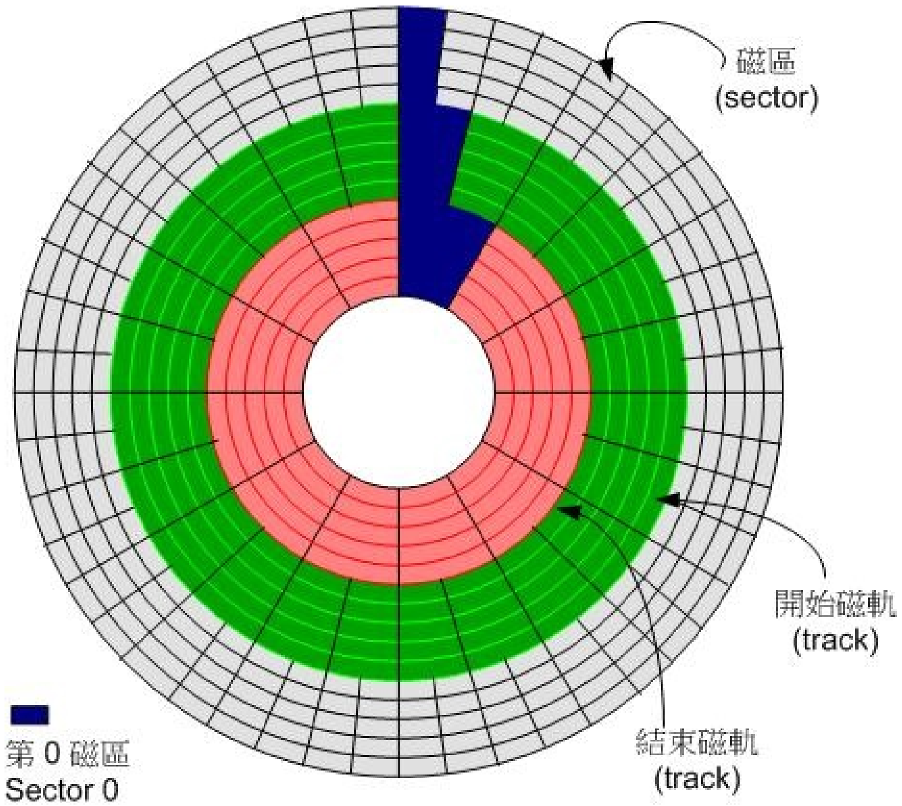


图0.2.5、盘片上的数据格式（图片取自维基百科）

由于盘片是圆的，并且通过机器手臂去读写数据，盘片要转动才能够让机器手臂读写。因此，通常数据写入当然就是以圆圈转圈的方式读写啰！ 所以，当初设计就是在类似盘片同心圆上面切出一个一个的小区块，这些小区块整合成一个圆形，让机器手臂上的磁头去存取。这个小区块就是磁盘的最小物理储存单位，称之为扇区 （sector），那同一个同心圆的扇区组合成的圆就是所谓的磁道（track）。 由于磁盘里面可能会有多个盘片，因此在所有盘片上面的同一个磁道可以组合成所谓的柱面 （cylinder）。

我们知道同心圆外圈的圆比较大，占用的面积比内圈多啊！所以，为了善用这些空间，因此外围的圆会具有更多的扇区[16]！ 就如同图 0.2.5 的示意一般。此外，当盘片转一圈时，外圈的扇区数量比较多，因此如果数据写入在外圈，转一圈能够读写的数据量当然比内圈还要多！ 因此通常数据的读写会由外圈开始往内写的喔！这是默认值啊！

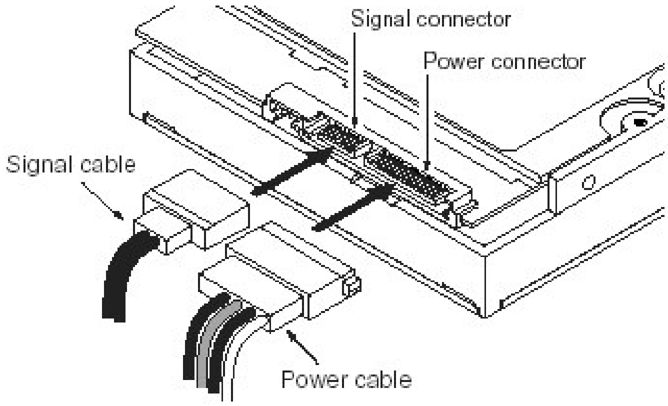
另外，原本硬盘的扇区都是设计成 512Byte 的容量，但因为近期以来硬盘的容量越来越大，为了减少数据量的拆解，所以新的大容量硬盘已经有 4KByte 的扇区设计！ 购买的时候也需要注意一下。也因为这个扇区的设计不同了，因此在磁盘的分区方面，目前有旧式的 MSDOS 相容模式，以及较新的 GPT 模式喔！ 在较新的 GPT 模式下，磁盘的分区通常使用扇区号码来设计，跟过去旧的 MSDOS 是通过柱面号码来分区的情况不同喔！相关的说明我们谈到磁盘管理 （第七章） 再来聊！

- 传输接口

为了要提升磁盘的传输速度，磁盘与主板的连接接口也经过多次的改版，因此有许多不同的接口喔！传统磁盘接口包括有 SATA, SAS, IDE 与 SCSI 等等。 若考虑外接式磁盘，那就还包括了 USB, eSATA 等等接口喔！不过目前 IDE 已经被 SATA 取代，而 SCSI 则被 SAS 取代，因此我们下面将仅介绍 SATA, USB 与 SAS 接口而已。

- SATA接口

如同华硕主板图示右下方所示为SATA硬盘的连接接口插槽。这种插槽所使用的排线比较窄小， 而且每个设备需要使用掉一条SATA线。因为SATA线比较窄小之故，所以对于安装与机箱内的通风都比较好！因此原本的IDE粗排线接口就被SATA取代了！ SATA的插槽示意图如下所示：



SATA cabling with separate power and signal attachments 图0.2.6、SATA 接口的排线 （图示取自 Seagate 网站）

由于SATA一条排线仅接一颗硬盘，所以你不需要调整跳针。不过一张主板上面SATA插槽的数量并不是固定的， 且每个插槽都有编号，在连接SATA硬盘与主板的时候，还是需要留意一下。此外，目前的 SATA 版本已经到了第三代 [17]， 每一代之间的传输速度如下所示，而且重点是，每一代都可以向下相容喔！只是速度上会差很多就是了。目前主流都是使用 SATA3 这个接口速度可达 600MByte/s 的接口！

版本	带宽 （Gbit/s）	速度 （MByte/s）
SATA 1.0	1.5	150
SATA 2.0	3	300
SATA 3.0	6	600

因为 SATA 传输接口传输时，通过的数据演算法的关系，当传输 10 位编码时，仅有 8 位为数据，其余 2 位为检验之用。因此带宽的计算上面， 使用的换算 （bit 转 Byte） 为 1:10 而不是 1Byte=8bits 喔！上表的对应要稍微注意一下。另外，虽然这个 SATA3 接口理论上可达 600MBytes/s 的传输速度， 不过目前传统的硬盘由于其物理组成的限制，一般极限速度大约在 150~200MByte/s 而已啦！所以厂商们才要发展固态硬盘啊！ ^_^

- SAS接口

早期工作站或大型大脑上面，为了读写速度与稳定性，因此在这样的机器上面，大多使用的是 SCSI 这种高阶的连接接口。 不过这种接口的速度后来被 SATA 打败了！但是 SCSI 有其值得开发的功能，因此后来就有串行式 SCSI （Serial Attached SCSI, SAS） 的发展。这种接口的速度比 SATA 来的快，而且连接的 SAS 硬盘的盘片转速与传输的速度也都比 SATA 硬盘好！ 只是...好贵喔！而且一般个人电脑的主板上面通常没有内置 SAS 连接接口，得要通过外接卡才能够支持。因此一般个人电脑主机还是以 SATA 接口为主要的磁盘连接接口啰。

版本	带宽 （Gbit/s）	速度 （MByte/s）
SAS 1	3	300
SAS 2	6	600
SAS 3	12	1200

因为这种接口的速度确实比较快喔！而且还支持例如热拔插等功能，因此，许多的设备连接会以这种接口来链接！ 例如我们经常会听到的磁盘阵列卡的连接插槽，就是利用这种 SAS 接口开发出来的支持的 SFF-8087 设备等等的 [18]。

- USB接口

如果你的磁盘是外接式的接口，那么很可能跟主板链接的就是 USB 这种接口了！这也是目前（2015） 最常见到的外接式磁盘接口了。 不过传统的 USB 速度挺慢的，即使是比较慢的传统硬盘，其传输率大概兜还有 80~120MBytes/s ，但传统的 USB 2.0 仅有大约 60MBytes/s 的理论传输率， 通常实做在主板上面的连接口，竟然都仅有 30~40 MByte/s 而已呢！实在发挥不出磁盘的性能啊！

为了改善 USB 的传输率，因此新一代的 USB 3.0 速度就快很多了！据说还有更新的 USB 3.1 正在发展中！这几代版本的带宽与速度制表如下 [19]：

版本	带宽 （Mbit/s）	速度 （MByte/s）
USB 1.0	12	1.5
USB 2.0	480	60
USB 3.0	5G	500
USB 3.1	10G	1000

跟 SATA 接口一样，不是理论速度到达该数值，实际上就可以跑到这么高！USB 3.0 虽然速度很快，但如果你去市面上面买 USB 的传统磁盘或闪存盘， 其实他的读写速度还是差不多在 100MBytes/s 而已啦！不过这样就超级快了！因为一般 USB2.0 的闪存盘读写速度大约是 40MBytes/10MBytes 左右而已说。 在购买这方面的外接式磁盘时，要特别考虑喔！

- 固态硬盘 （Solid State Disk, SSD）

传统硬盘有个很致命的问题，就是需要驱动马达去转动盘片～这会造成很严重的磁盘读取延迟！想想看，你得要知道数据在哪个扇区上面，然后再命令马达开始转， 之后再让磁头去读取正确的数据。另外，如果数据放置的比较离散（扇区分佈比较广又不连续），那么读写的速度就会延迟更明显！速度快不起来。因此， 后来就有厂商拿闪存去制作成大容量的设备，这些设备的连接接口也是通过 SATA 或 SAS，而且外型还做的跟传统磁盘一样！所以， 虽然这类的设备已经不能称为是磁盘 （因为没有磁头与盘片啊！都是内存！）。但是为了方便大家称呼，所以还是称为磁盘！只是跟传统磁盘 （Hard Disk Drive, HDD） 不同， 就称为固态硬盘 （Solid State Disk 或 Solid State Driver, SSD）。

固态硬盘最大的好处是，它没有马达不需要转动，而是通过内存直接读写的特性，因此除了没数据延迟且快速之外，还很省电！ 不过早期的 SSD 有个很重要的致命伤，就是这些闪存有“写入次数的限制”，因此通常 SSD 的寿命大概两年就顶天了！所以数据存放时， 需要考虑到备份或者是可能要使用 RAID 的机制来防止 SSD 的损毁[20]！



Tips SSD 真的好快！鸟哥曾经买过 Intel 较顶级的 SSD 来做过服务器的读取系统盘，然后使用类似 dd 的指令去看看读写的速度，竟然真的如同 intel 自己官网说的， 极速可以到达 500MBytes/s 哩！几乎就是 SATA3.0 的理论极限速度了！所以，近来在需要大量读取的环境中，鸟哥都是使用 SSD 阵列来处理！

其实我们在读写磁盘时，通常没有连续读写，大部分的情况下都是读写一大堆小文件，因此，你不要妄想传统磁盘一直转很少圈就可以读到所有的数据！ 通常很多小文件的读写，会很耗硬盘，因为盘片要转好多圈！这也很花人类的时间啊！SSD 就没有这个问题！也因为如此，近年来在测试磁盘的性能时， 有个很特殊的单位，称为每秒读写操作次数 （Input/Output Operations Per Second, IOPS）！这个数值越大，代表可操作次数较高，当然性能好的很！

- 选购与运行须知

如果你想要增加一颗硬盘在你的主机里头时，除了需要考虑你的主板可接受的插槽接口 （SATA/SAS）之外，还有什么要注意的呢？

- HDD 或 SSD 毕竟 HDD 与 SSD 的价格与容量真的差很多！不过，速度也差很多就是了！因此，目前大家的使用方式大多是这样的，使用 SSD 作为系统盘， 然后数据储存大多放置在 HDD 上面！这样系统运行快速 （SSD），而数据储存量也大 （HDD）。

- 容量 毕竟目前数据量越来越大，所以购买磁盘通常首先要考虑的就是容量的问题！目前 （2015）主流市场HDD容量已经到达 2TB以上， 甚至有的厂商已经生产高达 8TB 的产品呢！硬盘可能可以算是一种消耗品，要注意重要数据还是得常常备份出来喔！ 至于 SSD 方面，目前的容量大概还是在 128~256GB 之间吧！

- 缓冲内存 硬盘上头含有一个缓冲内存，这个内存主要可以将硬盘内常使用的数据高速缓存起来，以加速系统的读取性能。 通常这个缓冲内存越大越好，因为缓冲内存的速度要比数据从硬盘盘中被找出来要快的多了！ 目前主流的产品可达64MB左右的内存大小喔。
- 转速 因为硬盘主要是利用主轴马达转动盘片来存取，因此转速的快慢会影响到性能。 主流的台式机硬盘为每分钟7200转，笔记本电脑则是5400转。有的厂商也有推出高达10000转的硬盘， 若有高性能的数据存取需求，可以考虑购买高转速硬盘。
- 运行须知 由于硬盘内部机械手臂上的磁头与硬盘盘的接触是很细微的空间， 如果有抖动或者是脏污在磁头与硬盘盘之间就会造成数据的损毁或者是实体硬盘整个损毁～ 因此，正确的使用电脑的方式，应该是在电脑通电之后，就绝对不要移动主机，并免抖动到硬盘， 而导致整个硬盘数据发生问题啊！另外，也不要随便将插头拔掉就以为是顺利关机！因为机械手臂必须要归回原位， 所以使用操作系统的正常关机方式，才能够有比较好的硬盘保养啊！因为他会让硬盘的机械手臂归回原位啊！



Tips 可能因为环境的关系，电脑内部的风扇常常会卡灰尘而造成一些声响。很多朋友只要听到这种声响都是二话不说的 “用力拍几下机箱”就没有声音了～现在你知道了，这么做的后果常常就是你的硬盘容易坏掉！ 下次千万不要再这样做啰！

0.2.5 扩展卡与接口

你的服务器可能因为某些特殊的需求，因此需要使用主板之外的其他适配卡。所以主板上面通常会预留多个扩充接口的插槽， 这些插槽依据历史沿革，包括 PCI/AGP/PCI-X/PCIe 等等，但是由于 PCIe 速度快到太好用了，因此几乎所有的卡都以 PCIe 来设计了！ 但是有些比较老旧的卡可能还需要使用啊，因此一般主板大多还是会保留一两个 PCI 插槽，其他的则是以 PCIe 来设计。

由于各元件的价格直直落，现在主板上面通常已经整合了相当多的设备元件了！ 常见整合到主板的元件包括声卡、网卡、USB控制卡、显卡、磁盘阵列卡等等。 你可以在主板上面发现很多方形的芯片，那通常是一些个别的设备芯片喔。

不过，因为某些特殊的需求，有时你可能还是需要增加额外的扩展卡的。举例来说，我们如果需要一部个人电脑连接多个网域时（Linux 服务器用途）， 恐怕就得要有多个网卡。当你想要买网卡时，大卖场上面有好多耶！而且速度一样都是 giga 网卡 （Gbit/s），但价格差很多耶！ 观察规格，主要有 PCIe x1 以及 PCI 接口的！你要买哪种接口呢？

观察一下 0.2.3 显卡的章节内，你会发现到 PCI 接口的理论传输率最高指到 133MBytes/s 而已，而 PCIe 2.0 x1 就高达 500MBytes/s 的速度！ 鸟哥实测的结果也发现，PCI 接口的 giga 网卡极限速度大约只到 60MBytes/s 而已，而 PCIe 2.0 x1 的 giga 网卡确实可以到达大约 110MBytes/s 的速度！ 所以，购买设备时，还是要查清楚连接接口才行啦！

在 0.2.3 节也谈到 PCIe 有不同的信道数，基本上常见的就是 x1, x4, x8, x16 等，个人电脑主板常见是 x16 的，一般中阶服务器则大多有多个 x8 的接口， x16 反而比较少见。这些接口在主板上面的设计，主要是以插槽的长度来看的，例如华硕主板示意图中，左侧有 2 个 PCI 接口， 其他的则是 3 个 x16 的插槽，以及 2 个 x1 的插槽，看长度就知道了。

- 多信道卡 （例如 x8 的卡） 安装在少信道插槽 （例如 x4 的插槽） 的可用性

再回头看看图 0.2.1 的示意图，你可以发现 CPU 最多最多仅能支持 16 个 PCIe 3.0 的信道数，因此在图示当中就明白的告诉你， 你可以设计（1）一个 x16 （2）或者是两个 x8 ，（3）或者是两个 x4 加上一个 x8 的方式来增加扩展卡！这是可以直接链接到 CPU 的信道！咦！ 那为何图 0.2.2 可以有 3 个 x16 的插槽呢？原因是前两个属于 CPU 支持的，后面两个可能就是南桥提供的 PCIe 2.0 的接口了！ 那明明最多仅能支持一个 x16 的接口，怎么可能设计 3 个 x16 呢？

因为要让所有的扩展卡都可以安插在主板上面，所以在比较中高阶的主板上面，他们都会做出 x16 的插槽，但是该插槽内其实只有 x8 或 x4 的信道有用！ 其他的都是空的没有金手指（电路的意思）～咦！那如果我的 x16 的卡安装在 x16 的插槽，但是这个插槽仅有 x4 的电路设计，那我这张卡可以运行吗？ 当然可以！这就是 PCIe 的好处了！它可以让你这张卡仅使用 x4 的电路来传送数据，而不会无法使用！只是...你的这张卡的极限性能，就会只剩下 4/16 = 1/4 啰！

因为一般服务器惯用的扩展卡，大多数都使用 PCIe x8 的接口 （因为也没有什么设备可以将 PCIe 3.0 的 x8 速度用完啊！） ， 为了增加扩展卡的数量，因此服务器级的主板才会大多使用到 x8 的插槽说！反正，要发挥扩展卡的能力，就得要搭配相对应的插槽才行啦！



Tips 鸟哥近年来在搞小型云教室，为了加速需要有 10G 的网卡，这些网卡标准的接口为 PCIe 2.0 x8 的接口。有部主机上面需要安插这样的卡三张才行， 结果该主机上面仅有一个 x16，一个 x8 以及一个 x4 的 PCIe 接口，其中 x4 的那个接口使用的是 x8 的插槽，所以好佳在三张卡都可以安装在主板上面，且都可以运行！ 只是在极速运行时，实测的性能结果发现，那个安插在 x4 接口的网卡性能降很多！所以才会发现这些问题！提供给大家参考参考！

0.2.6 主板

这个小节我们特别再将主板拿出来说明一下，特别要讲的就是芯片组与扩展卡之间的关系了！

- 发挥扩展卡性能须考虑的插槽位置

如同图 0.2.1 所示，其实系统上面可能会有多个 x8 的插槽，那么到底你的卡插在哪个插槽上面性能最好？我们以该图来说，如果你是安插在左上方跟 CPU 直接连线的那几个插槽，那性能最佳！如果你是安插在左侧由上往下数的第五个 PCIe 2.0 x8 的插槽呢？那个插槽是与南桥连接，所以你的扩展卡数据需要先进入南桥跟大家抢带宽，之后要传向 CPU 时，还得要通过 CPU 与南桥的沟通管道，那条管道称为 DMI 2.0。

根据 Intel 方面的数据来看，DMI 2.0 的传输率是 4GT/s，换算成文件传输量时，大约仅有 2GByte/s 的速度，要知道，PCIe 2.0 x8 的理论速度已经达到 4GByte/s 了，但是与 CPU 的信道竟然仅有 2GB，性能的瓶颈就这样发生在 CPU 与南桥的沟通上面！因此，卡安装在哪个插槽上面，对性能而言也是影响很大的！所以插卡时，请详细阅读您主板上面的逻辑图示啊（类似本章的 Intel 芯片示意图）！尤其 CPU 与南桥沟通的带宽方面，特别重要喔！



Tips 因为鸟哥的 Linux 服务器，目前很多都需要执行一些虚拟化技术等会大量读写数据的服务，所以需要额外的磁盘阵列卡来提供数据的存放！同时得要提供 10G 网络让内部的多部服务器互相通过网络链接。过去没有这方面的经验时，扩展卡都随意乱插，反正能动就好！但实际分析过性能之后，哇！现在都不敢随便乱插了！性能差太多！每次在选购新的系统时，也都会优先去查看芯片逻辑图～确认性能瓶颈不会卡住在主板上，这才下手去购买！惨痛的经验产生惨痛的 $$ 飞走事件，所以，这里特别提出来跟大家分享的啦！

- 设备I/O位址与IRQ中断信道

主板是负责各个电脑元件之间的沟通，但是电脑元件实在太多了，有输出/输入/不同的储存设备等等，主板芯片组怎么知道如何负责沟通呐？这个时候就需要用到所谓的I/O位址与IRQ啰！

I/O位址有点类似每个设备的门牌号码，每个设备都有他自己的位址，一般来说，不能有两个设备使用同一个I/O位址，否则系统就会不晓得该如何运行这两个设备了。而除了I/O位址之外，还有个IRQ中断（Interrupt）这个咚咚。

如果I/O位址想成是各设备的门牌号码的话，那么IRQ就可以想成是各个门牌连接到邮件中心（CPU）的专门路径啰！各设备可以通过IRQ中断信道来告知CPU该设备的工作情况，以方便CPU进行工作分配的任务。老式的主板芯片组IRQ只有15个，如果你的周边接口太多时可能就会不够用，这个时候你可以选择将一些没有用到的周边接口关掉，以空出一些IRQ来给眞正需要使用的接口喔！当然，也有所谓的sharing IRQ的技术就是了！

- CMOS与BIOS

前面内存的地方我们有提过CMOS与BIOS的功能，在这里我们再来强调一下： CMOS主要的功能为记录主板上面的重要参数， 包括系统时间、CPU电压与频率、各项设备的I/O位址与IRQ等，由于这些数据的记录要花费电力，因此主板上面才有电池。 BIOS为写入到主板上某一块 flash 或 EEPROM 的程序，他可以在开机的时候执行，以载入CMOS当中的参数， 并尝试调用储存设备中的开机程序，进一步进入操作系统当中。BIOS程序也可以修改CMOS中的数据， 每种主板调用BIOS设置程序的按键都不同，一般台式机常见的是使用[del]按键进入BIOS设置画面。

- 连接周边设备的接口

主板与各项输出/输入设备的链接主要都是在主机机箱的后方，主要有：

- PS/2接口：这原本是常见的键盘与鼠标的接口，不过目前渐渐被USB接口取代，甚至较新的主板可能就不再提供 PS/2 接口了；
- USB接口：通常只剩下 USB 2.0 与 USB 3.0，为了方便区分，USB 3.0 为蓝色的插槽颜色喔！
- 声音输出、输入与麦克风：这个是一些圆形的插孔，而必须你的主板上面有内置音效芯片时，才会有这三个东西；
- RJ-45网络头：如果有内置网络芯片的话，那么就会有这种接头出现。 这种接头有点类似电话接头，不过内部有八蕊线喔！接上网络线后在这个接头上会有灯号亮起才对！
- HDMI：如果有内置显示芯片的话，可能就会提供这个与屏幕连接的接口了！这种接口可以同时传输声音与影像， 目前也是电视机屏幕的主流连接接口喔！

我们以华硕主板的链接接口来看的话，主要有这些：

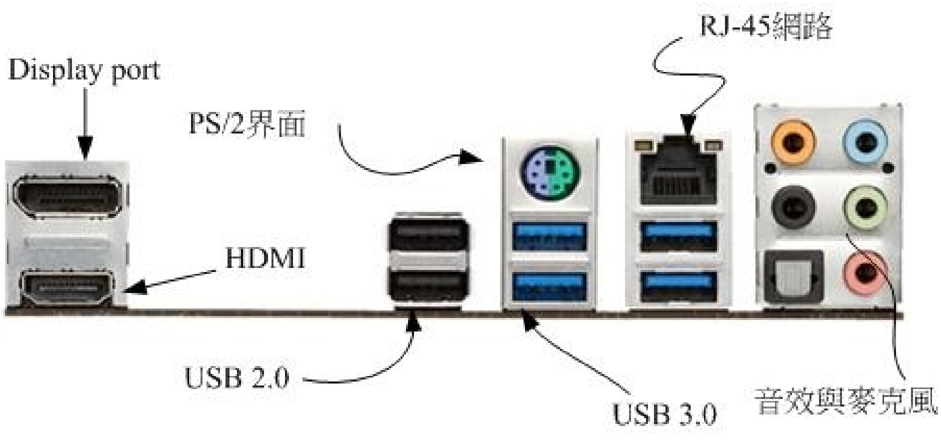


图0.2.7、连接周边接口

0.2.7 电源供应器

除了上面这些元件之外，其实还有一个很重要的元件也要来谈一谈，那就是电源供应器（Power）。 在你的机箱内，有个大大的铁盒子，上头有很多电源线会跑出来，那就是电源供应器了。 我们的CPU/RAM/主板/硬盘等等都需要用电，而近来的电脑元件耗电量越来越

高，以前很古早的230W电源已经不够用了， 有的系统甚至得要有500W以上的电源才能够运行～真可怕～

电源供应器的价差非常大！贵一点的300W可以到4000 NT，便宜一点的300W只要500 NT不到！ 怎么差这么多？没错～因为Power的用料不同，电源供应的稳定性也会差很多。如前所述，电源供应器相当于你的心脏， 心脏差的话，活动力就会不足了！所以， 稳定性差的电源供应器甚至是造成电脑不稳定的元凶呢！所以，尽量不要使用太差的电源供应器喔！

- 能源转换率

电源供应器本身也会吃掉一部份的电力的！如果你的主机系统需要 300W 的电力时，因为电源供应器本身也会消耗掉一部份的电力， 因此你最好要挑选400W以上的电源供应器。电源供应器出厂前会有一些测试数据，最好挑选高转换率的电源供应器。 所谓的高转换率指的是“输出的功率/输入的功率”。意思是说，假如你的主板用电量为250W， 但是电源供应器其实已经使用掉320W的电力，则转换率为：250/320=0.78的意思。 这个数值越高表示被电源供应器“玩掉”的电力越少，那就符合能源效益了！^_^

0.2.8 选购须知

在购买主机时应该需要进行整体的考虑，很难依照某一项标准来选购的。 老实说，如果你的公司需要一部服务器的话，建议不要自行组装，买品牌电脑的服务器比较好！ 这是因为自行组装的电脑虽然比较便宜，但是每项设备之间的适合性是否完美则有待自行检测。

另外，在性能方面并非仅考虑CPU的能力而已，速度的快慢与“整体系统的最慢的那个设备有关！”，如果你是使用最快速的Intel i7 系列产品，使用最快的 DDR3-1600 内存， 但是配上一个慢慢的过时显卡，那么整体的3D速度性能将会卡在那个显卡上面喔！所以，在购买整套系统时， 请特别留意需要全部的接口都考虑进去喔！尤其是当您想要升级时，要特别注意这个问题， 并非所有的旧的设备都适合继续使用的。

例题：你的系统使用 i7 的 4790 CPU，使用了 DDR3-1600 内存，使用了 PCIe 2.0 x8 的磁盘阵列卡，这张卡上面安装了 8 颗 3TB 的理论速度可达 200MByte/s 的硬盘 （假设为可加总速度的 RAID0 配置）， 是安插在 CPU 控制芯片相连的插槽中。网络使用 giga 网卡，安插在 PCIe 2.0 x1 的接口上。在这样的设备中，上述的哪个环节速度可能是你的瓶颈？答：

- DDR3-1600 的带宽可达：12.8GBytes/s
- 磁盘阵列卡理论传输率： PCIe 2.0 x8 为 4GBytes/s
- 磁盘每颗 200MBytes/s，共八颗，总效率为： 200MBytes*8 ~ 1.6GBytes/s
- 网络接口使用 PCIe 2.0 1x 所以接口速度可达 500MBytes/s，但是 Giga 网络最高为 125MBytes/s

通过上述分析，我们知道，速度最慢的为网络的 125MBytes/s ！所以，如果想要让整体性能提升，网络恐怕就是需要克服的一环！

- 系统不稳定的可能原因

除此之外，到底那个元件特别容易造成系统的不稳定呢？有几个常见的系统不稳定的状态是：

- 系统超频：这个行为很不好！不要这么做！
- 电源供应器不稳： 这也是个很严重的问题，当您测试完所有的元件都没有啥大问题时，记得测试一下电源供应器的稳定性！
- 内存无法负荷：现在的内存品质差很多，差一点的内存，可能会造成您的主机在忙碌的工作时， 产生不稳定或死机的现象喔！
- 系统过热：“热”是造成电子零件运行不良的主因之一，如果您的主机在夏天容易死机，冬天却还好，那么考虑一下加几个风扇吧！有助于机箱内的散热，系统会比较稳定喔！“这个问题也是很常见的系统死机的元凶！”（PS1:鸟哥之前的一台服务器老是容易死机，后来拆开机箱研究后才发现原来是北桥上面的小风扇坏掉了，导致北桥温度太高。后来换掉风扇就稳定多了。 PS2:还有一次整个实验室的网络都停了！检查了好久，才发现原来是网络交换器 switch 在夏天热到死机！后来只好用小电风扇一直吹他...）



Tips 事实上，要了解每个硬件的详细架构与构造是很难的！这里鸟哥仅是列出一些比较基本的概念而已。 另外，要知道某个硬件的制造商是哪间公司时，可以看该硬件上面的信息。 举例来说，主板上面都会列出这个主板的开发商与主板的型号，知道这两个信息就可以找到驱动程序了。 另外，显卡上面有个小小的芯片，上面也会列出显卡厂商与芯片信息喔。

0.3 数据表示方式

事实上我们的电脑只认识0与1，记录的数据也是只能记录0与1而已，所以电脑常用的数据是二进制的。 但是我们人类常用的数值运算是十进制，文字方面则有非常多的语言，台湾常用的语言就有英文、中文（又分正体与简体中文）、日文等。 那么电脑如何记录与显示这些数值/文字呢？就得要通过一系列的转换才可以啦！下面我们就来谈谈数值与文字的编码系统啰！

0.3.1 数字系统

早期的电脑使用的是利用通电与否的特性的眞空管，如果通电就是1，没有通电就是0， 后来沿用至今，我们称这种只有0/1的环境为二进制制，英文称为binary的哩。所谓的十进制指的是逢十进一位， 因此在个位数归为零而十位数写成1。所以所谓的二进制，就是逢二就前进一位的意思。

那二进制怎么用呢？我们先以十进制来解释好了。如果以十进制来说，3456的意义为：

$3456 = 3x10^3 + 4x10^2 + 5x10^1 + 6x10^0$

特别注意：“任何数值的零次方为1”所以100的结果就是1啰。 同样的，将这个原理带入二进制的环境中，我们来解释一下1101010的数值转为十进制的话，结果如下：

$1101010=1x2^6 + 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0 = 64 + 32 + 0x16 + 8 + 0x4 + 2 + 0x1 = 106$

这样你了解二进制的意义了吗？二进制是电脑基础中的基础喔！了解了二进制后，八进位、十六进制就依此类推啦！ 那么知道二进制转成十进制后，那如果有十进制数值转为二进制的环境时，该如何计算？ 刚刚是乘法，现在则是除法就对了！我们同样的使用十进制的106转成二进制来测试一下好了：

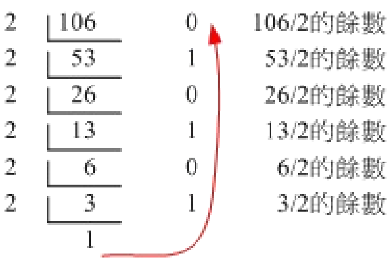


图0.3.1、十进制转二进制的方法

最后的写法就如同上面的红色箭头，由最后的数字向上写，因此可得到1101010的数字啰！ 这些数字的转换系统是非常重要的，因为电脑的加减乘除都是使用这些机制来处理的！ 有兴趣的朋友可以再参考一下其他计算计概论的书籍中，关于1的补数/2的补数等运算方式喔！

0.3.2 文字编码系统

既然电脑都只有记录0/1而已，甚至记录的数据都是使用Byte/bit等单位来记录的，那么文字该如何记录啊？ 事实上文字文件也是被记录为0与1而已，而这个文件的内容要被取出来查阅时，必须要经过一个编码系统的处理才行。 所谓的“编码系统”可以想成是一个“字码对照表”，他的概念有点像下面的图示：

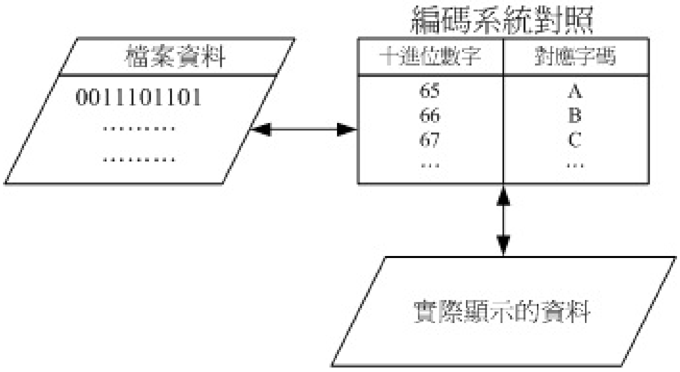


图0.3.2、数据参考编码表的示意图

当我们要写入文件的文字数据时，该文字数据会由编码对照表将该文字转成数字后，再存入文件当中。 同样的，当我们要将文件内容的数据读出时，也会经过编码对照表将该数字转成对应的文字后，再显示到屏幕上。 现在你知道为何浏览器上面如果编码写错时，会出现乱码了吗？这是因为编码对照表写错， 导致对照的文字产生误差之故啦！

常用的英文编码表为ASCII系统，这个编码系统中， 每个符号（英文、数字或符号等）都会占用1Bytes的记录， 因此总共会有28=256种变化。至于中文字当中的编码系统早期最常用的就是big5这个编码表了。 每个中文字会占用2Bytes，理论上最多可以有216=65536，亦即最多可达6万多个中文字。 但是因为big5编码系统并非将所有的位都拿来运用成为对照，所以并非可达这么多的中文字码的。 目前big5仅定义了一万三千多个中文字，很多中文利用big5是无法成功显示的～所以才会有造字程序说。

big5码的中文字编码对于某些数据库系统来说是很有问题的，某些字码例如“许、盖、功”等字， 由于这几个字的内部编码会被误判为单/双引号，在写入还不成问题，在读出数据的对照表时， 常常就会变成乱码。不只中文字，其他非英语系国家也常常会有这样的问题出现啊！

为了解决这个问题，由国际组织ISO/IEC跳出来制订了所谓的Unicode编码系统， 我们常常称呼的UTF8或万国码的编码就是这个咚咚。因为这个编码系统打破了所有国家的不同编码， 因此目前网际网络社会大多朝向这个编码系统在走，所以各位亲爱的朋友啊，记得将你的编码系统修订一下喔！

0.4 软件程序运行

鸟哥在上课时常常会开玩笑的问：“我们知道没有插电的电脑是一堆废铁，那么插了电的电脑是什么？” 答案是：“一堆会电人的废铁”！这是因为没有软件的运行，电脑的功能就无从发挥之故。 就好像没有了灵魂的躯体也不过就是行尸走肉，重点在于软件/灵魂啰！所以下面咱们就得要了解一下“软件”是什么。

一般来说，目前的电脑系统将软件分为两大类，一个是系统软件，一个是应用程序。但鸟哥认为我们还是得要了解一下什么是程序， 尤其是机器程序，了解了之后再来探讨一下为什么现今的电脑系统需要“操作系统”这玩意儿呢！

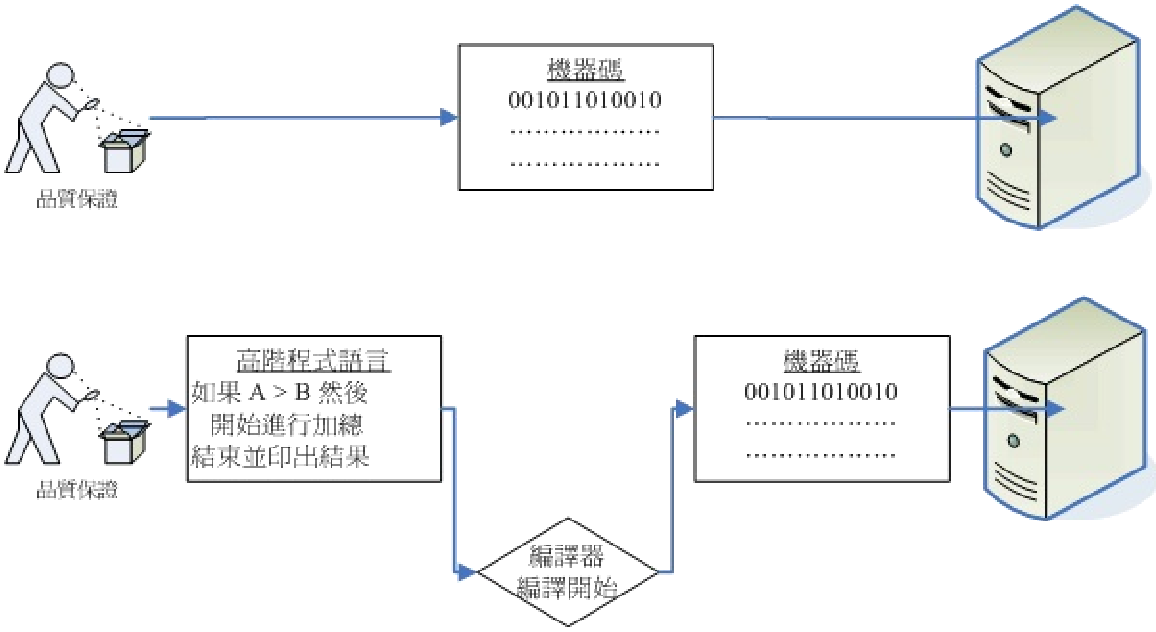
0.4.1 机器程序与编译程序

我们前面谈到电脑只认识0与1而已，而且电脑最重要的运算与逻辑判断是在CPU内部， 而CPU其实是具有微指令集的。因此，我们需要CPU帮忙工作时，就得要参考微指令集的内容， 然后撰写让CPU读的懂的指令码给CPU执行，这样就能够让CPU运行了。

不过这样的流程有几个很麻烦的地方，包括：

- 需要了解机器语言：机器只认识0与1，因此你必须要学习直接写给机器看的语言！ 这个地方相当的难呢！
- 需要了解所有硬件的相关功能函数：因为你的程序必须要写给机器看， 当然你就得要参考机器本身的功能，然后针对该功能去撰写程序码。例如，你要让DVD影片能够放映，那就得要参考DVD光驱的硬件信息才行。万一你的系统有比较冷门的硬件，光是参考技术手册可能会昏倒～
- 程序不具有可携性：每个CPU都有独特的微指令集，同样的，每个硬件都有其功能函数。 因此，你为A电脑写的程序，理论上是没有办法在B电脑上面运行的！而且程序码的修改非常困难！ 因为是机器码，并不是人类看的懂得程序语言啊！
- 程序具有专一性：因为这样的程序必须要针对硬件功能函数来撰写， 如果已经开发了一支浏览器程序，想要再开发文件管理程序时，还是得从头再参考硬件的功能函数来继续撰写， 每天都在和“硬件”挑战！可能需要天天喝蛮牛了！@_@

那怎么解决啊？为了解决这个问题，电脑科学家设计出一种让人类看的懂得程序语言， 然后创造一种“编译器”来将这些人类能够写的程序语言转译成为机器能看懂得机器码， 如此一来我们修改与撰写程序就变的容易多了！目前常见的编译器有C, C++, Java, Fortran等等。 机器语言与高阶程序语言的差别如下所示：



图

0.4.1、编译器的角色

从上面的图示我们可以看到高阶程序语言的程序码是很容易察看的！鸟哥已经将程序码（英文）写成中文说～ 这样比较好理解啦！所以这样已经将程序的修改问题处理完毕了。 问题是，在这样的环境下面我们还是得要考虑整体的硬件系统来设计程序喔！

举例来说，当你需要将运行的数据写入内存中，你就得要自行分配一个内存区块出来让自己的数据能够填上去， 所以你还得要了解到内存的位址是如何定位的，啊！眼泪还是不知不觉的流了下来... 怎么写程序这么麻烦啊！

为了要克服硬件方面老是需要重复撰写控制码的问题，所以就有操作系统（Operating System, OS）的出现了！ 什么是操作系统呢？下面就来谈一谈先！

0.4.2 操作系统

如同前面提到的，在早期想要让电脑执行程序就得要参考一堆硬件功能函数，并且学习机器语言才能够撰写程序。 同时每次写程序时都必须要重新改写，因为硬件与软件功能不见得都一致之故。那如果我能够将所有的硬件都驱动， 并且提供一个发展软件的参考接口来给工程师开发软件的话，那发展软件不就变的非常的简单了？那就是操作系统啦！

- 操作系统核心（Kernel）

操作系统（Operating System, OS）其实也是一组程序， 这组程序的重点在于管理电脑的所有活动以及驱动系统中的所有硬件。 我们刚刚谈到电脑没有软件只是一堆废铁，那么操作系统的功能就是让CPU可以开始判断逻辑与运算数值、 让内存可以开始载入/读出数据与程序码、让硬盘可以开始被存取、让网卡可以开始传输数据、 让所有周边可以开始运行等等。总之，硬件的所有动作都必须要通过这个操作系统来达成就是了。

上述的功能就是操作系统的核心（Kernel）了！你的电脑能不能做到某些事情，都与核心有关！只有核心有提供的功能，你的电脑系统才能帮你完成！举例来说，你的核心并不支持TCP/IP的网络协定，那么无论你购买了什么样的网卡，这个核心都无法提供网络能力的！

但是单有核心我们使用者也不知道能作啥事的～因为核心主要在管控硬件与提供相关的能力（例如存取硬盘、网络功能、CPU资源取得等），这些管理的动作是非常的重要的，如果使用者能够直接使用到核心的话，万一使用者不小心将核心程序停止或破坏，将会导致整个系统的崩溃！因此核心程序所放置到内存当中的区块是受保护的！并且开机后就一直常驻在内存当中。



Tips 所以整部系统只有核心的话，我们就只能看着已经准备好运行（Ready）的电脑系统，但无法操作他！好像有点望梅止渴的那种感觉啦！这个时候就需要软件的帮忙了！

- 系统调用（System Call）

既然我的硬件都是由核心管理，那么如果我想要开发软件的话，自然就得要去参考这个核心的相关功能！唔！如此一来不是从原本的参考硬件函数变成参考核心功能，还是很麻烦啊！有没有更简单的方法啊！

为了解决这个问题，操作系统通常会提供一整组的开发接口给工程师来开发软件！工程师只要遵守该开发接口那就很容易开发软件了！举例来说，我们学习C程序语言只要参考C程序语言的函数即可，不需要再去考虑其他核心的相关功能，因为核心的系统调用接口会主动的将C程序语言的相关语法转成核心可以了解的任务函数，那核心自然就能够顺利运行该程序了！

如果我们将整个电脑系统的相关软/硬件绘制成图的话，他的关系有点像这样：

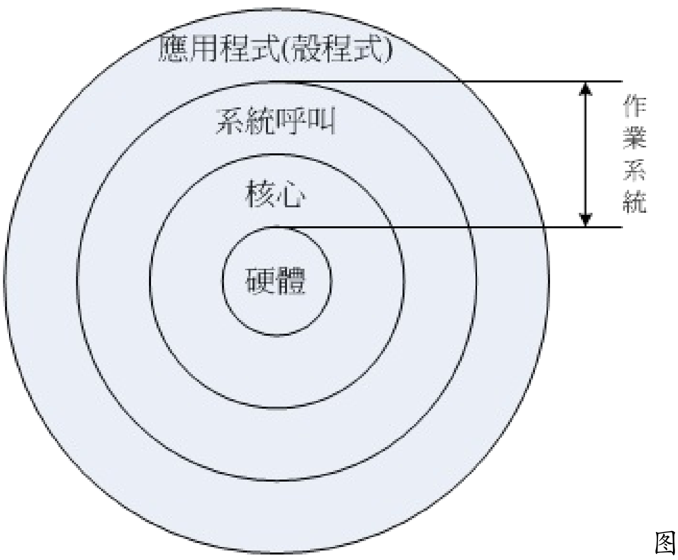


图0.4.2、操作系统的角色

电脑系统主要由硬件构成，然后核心程序主要在管理硬件，提供合理的电脑系统资源分配（包括CPU资源、内存使用资源等等）， 因此只要硬件不同（如x86架构与RISC架构的CPU），核心就得要进行修改才行。 而由于核心只会进行电脑系统的资源分配，所以在上头还需要有应用程序的提供，使用者才能够操作系统的。

为了保护核心，并且让程序设计师比较容易开发软件，因此操作系统除了核心程序之外，通常还会提供一整组开发接口， 那就是系统调用层。软件开发工程师只要遵循公认的系统调用参数来开发软件，该软件就能够在该核心上头运行。 所以你可以发现，软件与核心有比较大的关系，与硬件关系则不大！硬件也与核心有比较大的关系！ 至于与使用者有关的，那就是应用程序啦！



Tips 在定义上，只要能够让计算机硬件正确无误的运行，那就算是操作系统了。所以说， 操作系统其实就是核心与其提供的接口工具，不过就如同上面讲的，因为最阳春的核心缺乏了与使用者沟通的友好接口， 所以在目前，一般我们提到的“操作系统”都会包含核心与相关的使用者应用软件呢！

简单的说，上面的图示可以带给我们下面的概念：

- 操作系统的核心层直接参考硬件规格写成， 所以同一个操作系统程序不能够在不一样的硬件架构下运行。举例来说，个人电脑版的Windows 8.1 不能直接在 ARM 架构 （手机与平板硬件） 的电脑下运行。
- 操作系统只是在管理整个硬件资源，包括CPU、内存、输入输出设备及文件系统文件。 如果没有其他的应用程序辅助，操作系统只能让电脑主机准备妥当（Ready）而已！并无法运行其他功能。 所以你现在知道为何Windows上面要达成网页影像的运行还需要类似PhotoImpact或Photoshop之类的软件安装了吧？
- 应用程序的开发都是参考操作系统提供的开发接口， 所以该应用程序只能在该操作系统上面运行而已，不可以在其他操作系统上面运行的。 现在您知道为何去购买线上游戏的光盘时，光盘上面会明明白白的写着该软件适合用于哪一种操作系统上了吧？ 也该知道某些游戏为何不能够在Linux上面安装了吧？
- 核心功能

既然核心主要是在负责整个电脑系统相关的资源分配与管理，那我们知道其实整部电脑系统最重要的就是CPU与内存， 因此，核心至少也要有这些功能的：

- 系统调用接口（System call interface） 刚刚谈过了，这是为了方便程序开发者可以轻易的通过与核心的沟通，将硬件的资源进一步的利用， 于是需要有这个简易的接口来方便程序开发者。

- 程序管理（Process control） 总有听过所谓的“多任务环境”吧？一部电脑可能同时间有很多的工作跑到CPU等待运算处理， 核心这个时候必须要能够控制这些工作，让CPU的资源作有效的分配才行！另外， 良好的CPU调度机制（就是CPU先运行那个工作的排列顺序）将会有效的加快整体系统性能呢！

- 内存管理（Memory management） 控制整个系统的内存管理，这个内存控制是非常重要的，因为系统所有的程序码与数据都必须要先存放在内存当中。 通常核心会提供虚拟内存的功能，当内存不足时可以提供内存交换（swap）的功能哩。

- 文件系统管理（Filesystem management） 文件系统的管理，例如数据的输入输出（I/O）等等的工作啦！还有不同文件格式的支持啦等等， 如果你的核心不认识某个文件系统，那么您将无法使用该文件格式的文件啰！例如：Windows 98就不认识NTFS文件格式的硬盘；

- 设备的驱动（Device drivers） 就如同上面提到的，硬件的管理是核心的主要工作之一，当然啰，设备的驱动程序就是核心需要做的事情啦！ 好在目前都有所谓的“可载入模块”功能，可以将驱动程序编辑成模块，就不需要重新的编译核心啦！ 这个也会在后续的第十九章当中提到的！



Tips 事实上，驱动程序的提供应该是硬件厂商的事情！硬件厂商要推出硬件时，应该要自行参考操作系统的驱动程序开发接口， 开发完毕后将该驱动程序连同硬件一同贩卖给使用者才对！举例来说，当你购买显卡时， 显卡包装盒都会附上一片光盘，让你可以在进入Windows之后进行驱动程序的安装啊！

- 操作系统与驱动程序

老实说，驱动程序可以说是操作系统里面相当重要的一环了！不过，硬件可是持续在进步当中的！ 包括主板、显卡、硬盘等等。那么比较晚推出的较新的硬件，例如显卡，我们的操作系统当然就不认识啰！ 那操作系统该如何驱动这块新的显卡？为了克服这个问题，操作系统通常会提供一个开发接口给硬件开发商， 让他们可以根据这个接口设计可以驱动他们硬件的“驱动程序”，如此一来，只要使用者安装驱动程序后， 自然就可以在他们的操作系统上面驱动这块显卡了。

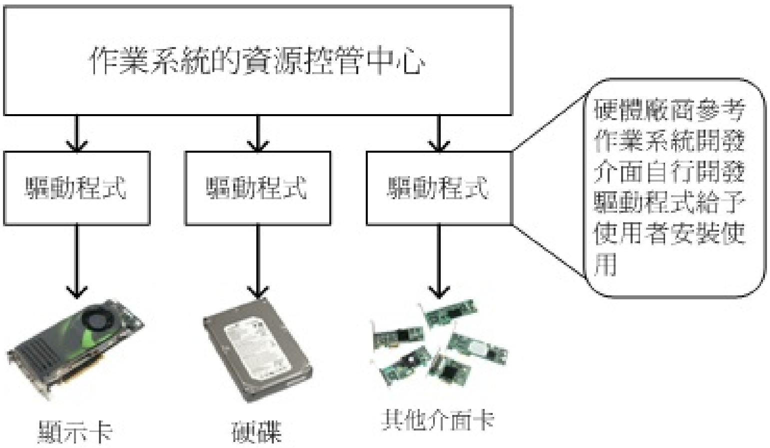


图0.4.3、驱动程序与操作系统的关系

由上图我们可以得到几个小重点：

- 操作系统必须要能够驱动硬件，如此应用程序才能够使用该硬件功能；
- 一般来说，操作系统会提供开发接口，让开发商制作他们的驱动程序；
- 要使用新硬件功能，必须要安装厂商提供的驱动程序才行；
- 驱动程序是由厂商提供的，与操作系统开发者无关。

所以，如果你想要在某个操作系统上面安装一张新的显卡，那么请要求该硬件厂商提供适当的驱动程序吧！ ^_^！ 为什么要强调“适当的驱动程序”呢？ 因为驱动程序仍然是依据操作系统而开发的， 所以，给Windows用的驱动程序当然不能使用于Linux的环境下了。

0.4.3 应用程序

应用程序是参考操作系统提供的开发接口所开发出来软件，这些软件可以让使用者操作，以达到某些电脑的功能利用。 举例来说，办公室软件（Office）主要是用来让使用者办公用的；影像处理软件主要是让使用者用来处理影音数据的； 浏览器软件主要是让使用者用来上网浏览用的等等。

需要注意的是，应用程序是与操作系统有关系的，如同上面的图示当中的说明喔。因此，如果你想要购买新软件， 请务必参考软件上面的说明，看看该软件是否能够支持你的操作系统啊！举例来说，如果你想要购买线上游戏光盘， 务必参考一下该光盘是否支持你的操作系统，例如是否支持Windows XP/Windows Vista/MAC/Linux等等。 不要购买了才发现该软件无法安装在你的操作系统上喔！

我们拿常见的微软公司的产品来说明。你知道Windows 8.1, Office 2013之间的关系了吗？

- Windows 8.1 是一套操作系统，他必须先安装到个人电脑上面，否则电脑无法开机运行；
- Windows 7 与Windows 8.1 是两套不同的操作系统，所以能在Win 7上安装的软件不见得可在Win 8.1上安装；
- Windows 8.1安装好后，就只能拥有很少的功能，并没有办公室软件；
- Office 2013是一套应用程序，要安装前必须要了解他能在哪些操作系统上面运行。

0.5 重点回顾

- 计算机的定义为："接受使用者输入指令与数据，经由中央处理器的数学与逻辑单元运算处理后，以产生或储存成有用的信息"；
- 电脑的五大单元包括：输入单元、输出单元、控制单元、算数逻辑单元、存储单元五大部分。其中CPU占有控制、算术逻辑单元，存储单元又包含内存与辅助内存；
- 数据会流进/流出内存是CPU所发布的控制命令，而CPU实际要处理的数据则完全来自于内存；
- CPU依设计理念主要分为：精简指令集（RISC）与复杂指令集（CISC）系统；
- 关于CPU的频率部分：外频指的是CPU与外部元件进行数据传输时的速度，倍频则是CPU内部用来加速工作性能的一个倍数， 两者相乘才是CPU的频率速度；
- 新的 CPU 设计中，已经将北桥的内存控制芯片整合到 CPU 内，而 CPU 与内存、显卡沟通的总线通常称为系统总线。 南桥就是所谓的输入输出（I/O）总线，主要在联系硬盘、USB、网卡等周边设备；
- CPU每次能够处理的数据量称为字组大小（word size），字组大小依据CPU的设计而有32位与64位。 我们现在所称的电脑是32或64位主要是依据这个 CPU解析的字组大小而来的！
- 个人电脑的内存主要元件为动态随机存取内存（Dynamic Random Access Memory, DRAM）， 至于CPU内部的第二层高速缓存则使用静态随机存取内存（Static Random Access Memory, SRAM）；
- BIOS（Basic Input Output System）是一套程序，这套程序是写死到主板上面的一个内存芯片中， 这个内存芯片在没有通电时也能够将数据记录下来，那就是只读存储器（Read Only Memory, ROM）；
- 目前主流的外接卡接口大多为 PCIe 接口，且最新为 PCIe 3.0，单信道速度高达 1GBytes/s
- 常见的显卡连接到屏幕的接口有 HDMI/DVI/D-Sub/Display port 等等。HDMI 可同时传送影像与声音。
- 传统硬盘的组成为：圆形盘片、机械手臂、 磁头与主轴马达所组成的，其中盘片的组成为扇区、磁道与柱面；
- 磁盘连接到主板的接口大多为 SATA 或 SAS，目前台式机主流为 SATA 3.0，理论极速可达 600MBytes/s。
- 常见的文字编码为 ASCII，繁体中文编码主要有 Big5 及 UTF8 两种，目前主流为 UTF8
- 操作系统（Operating System, OS）其实也是一组程序， 这组程序的重点在于管理电脑的所有活动以及驱动系统中的所有硬件。
- 电脑主要以二进制作为单位，常用的磁盘容量单位为Bytes，其单位换算为1 Byte = 8bits。
- 最阳春的操作系统仅在驱动与管理硬件，而要使用硬件时，就得需要通过应用软件或者是壳程序（shell）的功能， 来调用操作系统操纵硬件工作。目前称为操作系统的，除了上述功能外，通常已经包含了日常工作所需要的应用软件在内了。

0.6 本章习题

- 根据本章内文的说明，请找出目前全世界跑的最快的超级计算机的：（1）系统名称（2）所在位置（3）使用的 CPU 型号与规格（4）总共使用的 CPU 数量（5）全功率操作 1 天时，可能耗用的电费（请上台电网站查询相关电价来计算）。

- 动动手实作题：假设你不知道你的主机内部的各项元件数据，请拆开你的主机机箱，并将内部所有的元件拆开，并且依序列出：

  - CPU的厂牌、型号、最高频率；
  - 内存的容量、接口（DDR/DDR2/DDR3 等）；
  - 显卡的接口（AGP/PCIe/内置）与容量
  - 主板的厂牌、南北桥的芯片型号、BIOS的厂牌、有无内置的网卡或声卡等
  - 硬盘的连接接口（SATA/SAS等）、硬盘容量、转速、缓冲内存容量等。然后再将他组装回去。注意，折装前务必先取得你主板的说明书，因此你可能必须要上网查询上述的各项数据。
- 利用软件：假设你不想要拆开主机机箱，但想了解你的主机内部各元件的信息时，该如何是好？如果使用的是Windows操作系统，可使用CPU-Z（http://www.cpuid.com/cpuz.php）这套软件，如果是Linux环境下，可以使用“cat /proc/cpuinfo” 及使用“lspci”来查阅各项元件的型号；

- 如本章图0.2.1所示，找出第四代 Intel i7 4790 CPU 的：（1）与南桥沟通的 DMI 带宽有多大？（2）第二层高速缓存的容量多大？（3）最大 PCIe 信道数量有多少？并据以说明主板上面 PCIe 插槽的数量限制。（请 google 此 CPU 相关数据即可发现）

- 由 google 查询 Intel SSD 520 固态硬盘相关的功能表，了解（1）连接接口、（2）最大读写速度及（3）最大随机读写数据（IOPS）的数据。

0.7 参考资料与延伸阅读

- [1]名片型电脑，或单版电脑：
  - 香蕉派台湾官网：http://tw.bananapi.org/
  - Xapple pi 粉丝团：https://www.facebook.com/roseapplepi
- [2]可穿戴式电脑：http://en.wikipedia.org/wiki/Wearable_computer
- [3]对于CPU的原理有兴趣的读者，可以参考维基百科的说明： 英文 CPU（http://en.wikipedia.org/wiki/CPU） 中文CPU（http://zh.wikipedia.org/wiki/中央处理器）。
- [4]图片参考： Wiki book: http://en.wikibooks.org/wiki/IB/Group_4/Computer_Science/Computer_Organisation 作者：陈锦辉，“计算机概论-探索未来2008”，金禾信息，2007出版
- [5]更详细的RISC架构可以参考维基百科： http://zh.wikipedia.org/w/index.php?title=精简指令集&variant=zh-cn 相关的 CPU 种类可以参考： Oracle SPARC: http://en.wikipedia.org/wiki/SPARC IBM Power CPU: http://en.wikipedia.org/wiki/IBM_POWER_microprocessors
- [6]关于ARM架构的说明，可以参考维基百科： http://zh.wikipedia.org/w/index.php?title=ARM架构&variant=zh-cn
- [7]更详细的CISC架构可参考维基百科： http://zh.wikipedia.org/w/index.php?title=CISC&variant=zh-cn
- [8]更详细的x86架构发展史可以参考维基百科： http://zh.wikipedia.org/w/index.php?title=X86&variant=zh-cn
- [9]用来观察 CPU 相关信息的 CPU-Z 软件网站： http://www.cpuid.com/softwares/cpu-z.html
- [10]Intel i7 4790 CPU 的详细规格介绍 http://ark.intel.com/zh-cn/products/80806/Intel-Core-i7-4790-Processor-8M-Cache-up-to-4_00-GHz
- [11]DDR 内存的详细规格介绍 http://zh.wikipedia.org/wiki/DDR_SDRAM
- [12]相关的固件说明可参考维基百科： http://zh.wikipedia.org/w/index.php?title=固件&variant=zh-hant
- [13]相关 EEPROM 可以参考维基百科： http://zh.wikipedia.org/w/index.php?title=EEPROM&variant=zh-cn
- [14]相关 BIOS 的说明可以参考维基百科： http://zh.wikipedia.org/w/index.php?title=BIOS&variant=zh-cn
- [15]相关 PCIe 的说明可以参考维基百科： http://en.wikipedia.org/wiki/PCI_Express
- [16]关于盘片数据的说明：Zone bit recording ： http://en.wikipedia.org/wiki/Zone_bit_recording
- [17]关于 SATA 磁盘接口的 wiki 说明 ： http://zh.wikipedia.org/wiki/SATA
- [18]关于 SAS 磁盘接口的 wiki 说明 ： http://en.wikipedia.org/wiki/SCSI#SCSI-EXPRESS http://en.wikipedia.org/wiki/Serial_attached_SCSI

- [19]关于 USB 接口的 wiki 说明 ： http://en.wikipedia.org/wiki/USB
- [20]关于 SSD 的 wiki 说明 ： http://en.wikipedia.org/wiki/Solid-state_drive

- 感谢：本章当中出现很多图示，很多是从Tom's Hardware（http://www.tomshardware.com/）网站取得的，在此特别感谢！

2008/07/22：利用暑假期间足足写了快要两个星期这篇才写完！好多图示都不知道如何呈现比较漂亮～@_@ 2008/07/29：又加入了SATA/IDE的连线排线，还有一些额外的图示。2009/08/03：加入电源供应器是心脏一词的说明 2009/08/03：更正原本 BIOS 只放于 ROM 的数据，新的 BIOS 通常放于 EEPROM 或 Flash 内存中。 2010/10/19：感谢讨论区网友 186003415a 兄的回报，发现 DDR II 的外频写错了！是 200MHz 才对喔！ 2015/04/16：旧版针对 CentOS5 时所作的旧版文章：旧的计概内容 2015/04/16：删除了一些较旧的数据，如 IDE，增加了不少新的东西！并感谢讨论区 littlebat 兄的意见，修改了不少的磁盘数据

第一章、Linux是什么与如何学习

最近更新日期：20//

众所皆知的，Linux的核心原型是1991年由托瓦兹（Linus Torvalds）写出来的，但是托瓦兹为何可以写出Linux这个操作系统？ 为什么他要选择386的计算机来开发？为什么Linux的发展可以这么迅速？又为什么Linux是免费且可以自由学习的？ 以及目前为何有这么多的Linux套件版本（distributions）呢？了解这些东西后，才能够知道为何Linux可以免除专利软件之争，并且了解到Linux为何可以同时在个人计算机与大型主机上面大放异彩！ 所以，在实际进入Linux的世界前，就让我们来谈一谈这些有趣的历史故事吧！ ^_^

1.1 Linux是什么

我们知道Linux这玩意儿是在计算机上面运行的，所以说Linux就是一组软件。问题是这个软件是操作系统还是应用程序？ 且Linux可以在哪些种类的计算机硬件上面运行？而Linux源自哪里？为什么 Linux 还不用钱？这些我们都得来谈一谈先！免得下次人家问你， 为什么复制软件不会违法时，你会答不出来啊！^_^

1.1.1 Linux是什么？操作系统/应用程序？

我们在第零章、计算机概论里面有提到过整个计算机系统的概念， 计算机主机是由一堆硬件所组成的，为了有效率的控制这些硬件资源，于是乎就有操作系统的产生了。 操作系统除了有效率的控制这些硬件资源的分配，并提供计算机运行所需要的功能（如网络功能）之外，为了要提供程序设计师更容易开发软件的环境，所以操作系统也会提供一整组系统调用接口来给软件设计师开发用喔！

知道为什么要讲这些了吗？嘿嘿！没错，因为Linux就是一套操作系统！如同下图所示，Linux就是核心与系统调用接口那两层。至于应用程序算不算Linux呢？当然不算啦！这点要特别注意喔！

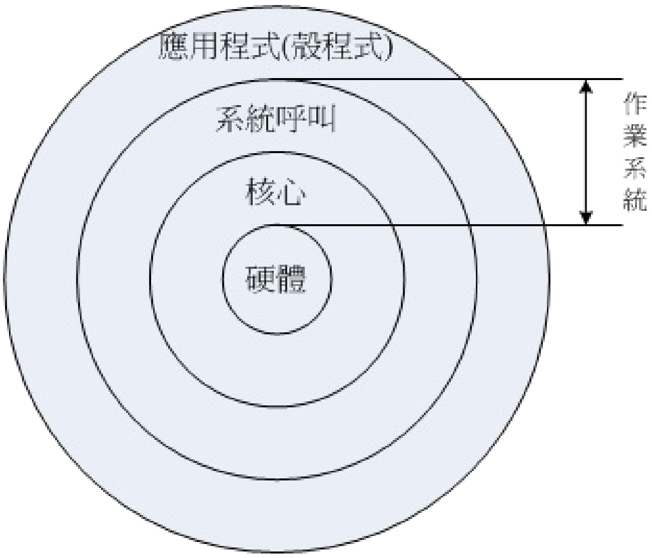


图1.1.1、操作系统的角色

由上图中我们可以看到其实核心与硬件的关系非常的强烈。早期的Linux是针对386来开发的， 由于Linux只是一套操作系统并不含有其他的应用程序，因此很多工程师在下载了 Linux 核心并且实际安装之后，就只能看着计算机开始运行了！接下来这些高级工程师为了自己的需求，再在Linux上面安装他们所需要的软件就是了。



Tips Torvalds先生在 1991 年写出Linux 核心的时候，其实该核心仅能“驱动386所有的硬件”而已， 所谓的“让386计算机开始运行，并且等待使用者指令输入”而已，事实上，当时能够在Linux上面跑的软件还很少呢！

由于不同的硬件他的功能函数并不相同，例如IBM的Power CPU与Intel的x86架构就是不一样！ 所以同一套操作系统是无法在不同的硬件平台上面运行的！举例来说，如果你想要让x86上面跑的那套操作系统也能够在Power CPU上运行时，就得要将该操作系统进行修改才行。 如果能够参考硬件的功能函数并据以修改你的操作系统程序码， 那经过改版后的操作系统就能够在另一个硬件平台上面运行了。 这个动作我们通常就称为“软件移植”了！

例题：请问Windows操作系统能否在苹果公司的MAC计算机上面安装与运行？答：由上面的说明中，我们知道硬件是由“核心”来控制的，而每种操作系统都有他自己的核心。 在2006年以前的苹果计算机公司是请IBM公司帮忙开发硬件（所谓的Power CPU）， 而苹果计算机公司则在该硬件架构上发展自家的操作系统（就是俗称的MAC是也）。Windows则是开发在x86架构上的操作系统之一， 因此Windows是没有办法安装到MAC计算机硬件上面的。

不过，在2006年以后，苹果计算机转而请Intel设计其硬件架构，亦即其硬件架构已经转为x86系统， 因此在2006年以后的苹果计算机若使用x86架构时，其硬件则“可能”可以安装Windows操作系统了。 不过，你可能需要自己想些方法来处理该硬件的兼容性啰！



Tips Windows操作系统本来就是针对个人计算机x86架构的硬件去设计的，所以他当然只能在x86的个人计算机上面运行， 在不同的硬件平台当然就无法运行了。也就是说，每种操作系统都是在他专门的硬件机器上面运行的喔！这点得要先了解。 不过，Linux由于是Open Source的操作系统，所以他的程序码可以被修改成适合在各种机器上面运行的， 也就是说，Linux是具有“可移植性”，这可是很重要的一个功能喔！ ^_^

Linux提供了一个完整的操作系统当中最底层的硬件控制与资源管理的完整架构， 这个架构是沿袭Unix良好的传统来的，所以相当的稳定而功能强大！此外， 由于这个优良的架构可以在目前的个人计算机（x86系统）上面跑， 所以很多的软件开发者渐渐的将他们的工作心血移转到这个架构上面，所以 Linux 操作系统也有很多的应用软件啦！

虽然Linux仅是其核心与核心提供的工具，不过由于核心、核心工具与这些软件开发者提供的软件的整合， 使得Linux成为一个更完整的、功能强大的操作系统啰！约略了解Linux是何物之后，接下来，我们要谈一谈，“为什么说Linux是很稳定的操作系统呢？他是如何来的？”

1.1.2 Linux之前，Unix的历史

早在Linux出现之前的二十年（大约在1970 年代），就有一个相当稳定而成熟的操作系统存在了！ 那就是Linux的老大哥“Unix”是也！怎么这么说呢？他们这两个家伙有什么关系呀？这里就给他说一说啰！

众所皆知的，Linux的核心是由Linus Torvalds在1991年的时候给他开发出来的， 并且丢到网络上提供大家下载，后来大家觉得这个小东西（Linux Kernel）相当的小而精巧， 所以慢慢的就有相当多的朋友投入这个小东西的研究领域里面去了！ 但是为什么这个小东西这么棒呢？又为什么大家都可以免费的下载这个东西呢？ 嗯！等鸟哥慢慢的唬xx....喔不！听我慢慢的道来！

- 1969年以前：一个伟大的梦想--Bell,MIT与GE的“Multics”系统

早期的计算机并不像现在的个人计算机一样普遍，他可不是一般人碰的起的呢～ 除非是军事或者是高科技用途，或者是学术单位的前瞻性研究，否则真的很难接触到。 非但如此，早期的计算机架构还很难使用，除了运算速度并不快之外，操作接口也很困扰的！ 因为那个时候的输入设备只有读卡机、输出设备只有打印机， 使用者也无法与操作系统互动（批次型操作系统）。

在那个时候，写程序是件很可怜的事情，因为程序设计者，必须要将程序相关的信息在读卡纸上面打洞， 然后再将读卡纸插入读卡机来将信息读入主机中运算。光是这样就很麻烦了，如果程序有个小地方写错， 哈哈！光是重新打卡就很惨，加上主机少，使用者众多，光是等待，就耗去很多的时间了！

在那之后，由于硬件与操作系统的改良，使得后来可以使用键盘来进行信息的输入。 不过，在一间学校里面，主机毕竟可能只有一部，如果多人等待使用，那怎么办？大家还是得要等待啊！ 好在1960年代初期麻省理工学院（MIT）发展了所谓的：“相容分时系统（Compatible Time-Sharing System, CTSS）”， 它可以让大型主机通过提供数个终端机（terminal）以连线进入主机，来利用主机的资源进行运算工作。 架构有点像这样：

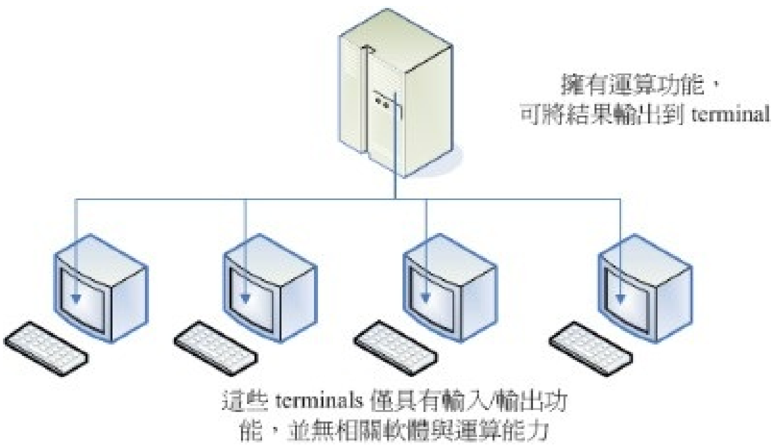


图1.1.2、早期主机与终端机的相关性图示



Tips 这个相容分时系统可以说是近代操作系统的始祖呢！他可以让多个使用者在某一段时间内分别使用CPU的资源， 感觉上你会觉得大家是同时使用该主机的资源！事实上，是CPU在每个使用者的工作之间进行切换， 在当时，这可是个划时代的技术喔！

如此一来，无论主机在哪里，只要在终端机前面进行输入输出的作业，就可利用主机提供的功能了。 不过，需要注意的是，此时终端机只具有输入/输出的功能，本身完全不具任何运算或者软件安装的能力。 而且，比较先进的主机大概也只能提供30个不到的终端机而已。

为了更加强化大型主机的功能，以让主机的资源可以提供更多使用者来利用，所以在1965年前后， 由贝尔实验室（Bell）、麻省理工学院（MIT）及奇异公司（GE, 或称为通用电器）共同发起了Multics的计划 [1]， Multics计划的目的是想要让大型主机可以达成提供300个以上的终端机连线使用的目标。 不过，到了1969年前后，计划进度落后，资金也短缺，所以该计划虽然继续在研究，但贝尔实验室还是退出了该计划的研究工作。 （Multics有复杂、多数的意思存在。）



Tips 最终Multics还是有成功的发展出他们的系统，完整的历史说明可以参考：http://www.multicians.org/网站内容。 Multics计划虽然后来没有受到很大的重视，但是他培养出来的人材是相当优秀的！ ^_^

- 1969年：Ken Thompson的小型file server system

在认为Multics计划不可能成功之后，贝尔研究室就退出该计划。不过，原本参与Multics计划的人员中，已经从该计划当中获得一些点子， Ken Thompson [2] 就是其中一位！

Thompson因为自己的需要，希望开发一个小小的操作系统以提供自己的需求。 在开发时，有一部DEC （Digital Equipment Corporation）公司推出的PDP-7刚好没人使用， 于是他就准备针对这部主机进行操作系统核心程序的撰写。本来Thompson应该是没时间的（有家有小孩的宿命？）， 无巧不巧的是，在1969年八月份左右，刚好Thompson的妻儿去了美西探亲，于是他有了额外的一个月的时间好好的待在家将一些构想实现出来！

经过四个星期的奋斗，他终于以组合语言（Assembler）写出了一组核心程序，同时包括一些核心工具程序， 以及一个小小的文件系统。那个系统就是Unix的原型！ 当时Thompson将Multics庞大的复杂系统简化了不少，于是同实验室的朋友都戏称这个系统为：Unics。（当时尚未有Unix的名称）

Thompson 的这个文件系统有两个重要的概念，分别是：

- 所有的程序或系统设备都是文件
- 不管建构编辑器还是附属文件，所写的程序只有一个目的，且要有效的完成目标。

这些概念在后来对于Linux的发展有相当重要的影响喔！



Tips 套一句常听到的广告词：“科技始终来自于人性”，当初Thompson会写这套Unix核心程序， 却是想要移植一套名为“太空旅游”的游戏呢！ ^_^

- 1973年：Unix的正式诞生，Ritchie等人以C语言写出第一个正式Unix核心

由于Thompson写的那个操作系统实在太好用了，所以在贝尔实验室内部广为流传，并且数度经过改版。 但是因为Unics本来是以组合语言写成的，而如第零章计算机概论谈到的， 组合语言具有专一性，加上当时的计算机机器架构都不太相同，所以每次要安装到不同的机器都得要重新编写组合语言，真不方便！

后来Thompson与Ritchie合作想将Unics改以高阶程序语言来撰写。当时现成的高阶程序语言有B语言。 但是由B语言所编译出来的核心性能不是很好。后来Dennis Ritchie [3] 将B语言重新改写成C语言，再以C语言重新改写与编译Unics的核心， 最后正名与发行出Unix的正式版本！



Tips 这群高级骇客实在很厉害！因为自己的需求来开发出这么多好用的工具！ C程序语言开发成功后，甚至一直沿用至今呢！你说厉不厉害啊！这个故事也告诉我们，不要小看自己的潜能喔！ 你想作的，但是现实生活中没有的，就动手自己搞一个来玩玩吧！

由于贝尔实验室是隶属于美国电信大厂AT&T公司的， 只是AT&T当时忙于其他商业活动，对于Unix并不支持也不排斥。此外，Unix在这个时期的发展者都是贝尔实验室的工程师， 这些工程师对于程序当然相当有研究，所以，Unix在此时当然是不容易被一般人所接受的！不过对于学术界的学者来说， 这个 Unix 真是学者们进行研究的福音！因为程序码可改写并且可作为学术研究之用嘛！

需要特别强调的是，由于Unix是以较高阶的C语言写的，相对于组合语言需要与硬件有密切的配合， 高阶的C语言与硬件的相关性就没有这么大了！所以，这个改变也使得Unix很容易被移植到不同的机器上面喔！

- 1977年：重要的Unix分支--BSD的诞生

虽然贝尔属于AT&T，但是AT&T此时对于Unix是采取较开放的态度，此外，Unix是以高阶的C语言写成的， 理论上是具有可移植性的！亦即只要取得Unix的源代码，并且针对大型主机的特性加以修订原有的源代码（Source Code）， 就可能将Unix移植到另一部不同的主机上头了。所以在1973年以后，Unix便得以与学术界合作开发！ 最重要的接触就是与加州柏克莱（Berkeley）大学的合作了。

柏克莱大学的Bill Joy [4]在取得了Unix的核心源代码后，着手修改成适合自己机器的版本， 并且同时增加了很多工具软件与编译程序，最终将它命名为Berkeley Software Distribution（BSD）。这个BSD是Unix很重要的一个分支，Bill Joy也是Unix业者“Sun（升阳）”这家公司的创办者！ Sun公司即是以BSD发展的核心进行自己的商业Unix版本的发展的。 （后来可以安装在x86硬件架构上面FreeBSD即是BSD改版而来！）

- 1979年：重要的 System V 架构与版权宣告

由于Unix的高度可移植性与强大的性能，加上当时并没有版权的纠纷， 所以让很多商业公司开始了Unix操作系统的发展，例如AT&T自家的System V、IBM的AIX以及HP与DEC等公司，都有推出自家的主机搭配自己的Unix操作系统。

但是，如同我们前面提到的，操作系统的核心（Kernel）必须要跟硬件配合， 以提供及控制硬件的资源进行良好的工作！而在早期每一家生产计算机硬件的公司还没有所谓的“协定”的概念， 所以每一个计算机公司出产的硬件自然就不相同啰！因此他们必须要为自己的计算机硬件开发合适的Unix系统。 例如在学术机构相当有名的Sun、Cray与HP就是这一种情况。 他们开发出来的Unix操作系统以及内含的相关软件并没有办法在其他的硬件架构下工作的！ 另外，由于没有厂商针对个人计算机设计Unix系统，因此，在早期并没有支持个人计算机的Unix操作系统的出现。



Tips 如同相容分时系统的功能一般，Unix强调的是多用户多任务的环境！ 但早期的286个人计算机架构下的CPU是没有能力达到多任务的作业，因此，并没有人对移植Unix到x86的计算机上有兴趣。

每一家公司自己出的Unix虽然在架构上面大同小异，但是却真的仅能支持自身的硬件， 所以啰，早先的Unix只能与服务器（Server）或者是大型工作站（Workstation）划上等号！ 但到了 1979 年时，AT&T推出 System V 第七版 Unix 后，这个情况就有点改善了。 这一版最重要的特色是可以支持x86架构的个人计算机系统，也就是说 System V 可以在个人计算机上面安装与运行了。

不过因为AT&T由于商业的考虑，以及在当时现实环境下的思考，于是想将Unix的版权收回去。因此， AT&T在1979年发行的第七版Unix中，特别提到了 “不可对学生提供源代码”的严格限制！ 同时，也造成Unix业界之间的紧张气氛，并且也引爆了很多的商业纠纷～



Tips 目前被称为纯种的Unix指的就是System V以及BSD这两套啰！

- 1984年之一：x86架构的Minix操作系统开始撰写并于两年后诞生

关于1979年的版权声明中，影响最大的当然就是学校教Unix核心源代码相关学问的教授了！想一想，如果没有核心源代码，那么如何教导学生认识Unix呢？这问题对于 Andrew Tanenbaum （谭宁邦, [5]）教授来说，实在是很伤脑筋的！不过，学校的课程还是得继续啊！那怎么办？

既然1979年的Unix第七版可以在Intel的x86架构上面进行移植， 那么是否意味着可以将Unix改写并移植到x86上面了呢？在这个想法上， 谭宁邦教授于是乎自己动手写了Minix这个Unix Like的核心程序！ 在撰写的过程中，为了避免版权纠纷，谭宁邦完全不看Unix核心源代码！并且强调他的Minix必须能够与Unix相容才行！谭宁邦在1984年开始撰写核心程序， 到了1986年终于完成，并于次年出版Minix相关书籍，同时与新闻群组（BBS及News）相结合～



Tips 之所以称为Minix的原因，是因为他是个Mini （微小的） 的Unix系统啰！^_^

这个Minix版本比较有趣的地方是，他并不是完全免费的，无法在网络上提供下载！ 必须要通过磁片/磁带购买才行！虽然真的很便宜～不过，毕竟因为没有在网络上流传， 所以Minix的传递速度并没有很快速！此外，购买时，随磁片还会附上Minix的源代码！ 这意味着使用者可以学习Minix的核心程序设计概念喔！ （这个特色对于Linux的启始开发阶段，可是有很大的关系喔！）

此外，Minix操作系统的开发者仅有谭宁邦教授，因为学者很忙啊 （鸟哥当了老师之后，才发现，真的忙...）！加上谭宁邦始终认为Minix主要用在教育用途上面， 所以对于Minix是点到为止！没错，Minix是很受欢迎，不过，使用者的要求/需求的声音可能就比较没有办法上升到比较高的地方了！ 这样说，你明白吧？^_^

- 1984年之二：GNU计划与FSF基金会的成立

Richard Mathew Stallman（史托曼）在1984年发起的GNU计划，对于现今的自由软件风潮，真有不可磨灭的地位！目前我们所使用的很多自由软件或开源软件，几乎均直接或间接受益于GNU这个计划呢！ 那么史托曼是何许人也？为何他会发起这个GNU计划呢？

- 一个分享的环境：

Richard Mathew Stallman（生于1953年， 网络上自称的ID为RMS, [6]）从小就很聪明！他在1971年的时候，进入骇客圈中相当出名的人工智能实验室（AI Lab.）， 这个时候的骇客专指计算机功力很强的人，而非破坏计算机的怪客（cracker）喔！

当时的骇客圈对于软件的着眼点几乎都是在"分享"，骇客们都认为互相学习对方的程序码，这样才是产生更优秀的程序码的最佳方式！ 所以 AI 实验室的骇客们通常会将自己的程序码公布出来跟大家讨论喔！这个特色对于史托曼的影响很大！

不过，后来由于管理阶层以及骇客群们自己的生涯规划等问题，导致实验室的优秀骇客离开该实验室， 并且进入其他商业公司继续发展优秀的软件。但史托曼并不服输，仍然持续在原来的实验室开发新的程序与软件。 后来，他发现到，自己一个人并无法完成所有的工作，于是想要成立一个开放的团体来共同努力！

- 使用Unix开发阶段：

1983年以后，因为实验室硬件的更换，使得史托曼无法继续以原有的硬件与操作系统继续自由程序的撰写～ 而且他进一步发现到，过去他所使用的Lisp操作系统，是麻省理工学院的专利软件， 是无法共享的，这对于想要成立一个开放团体的史托曼是个阻碍。于是他便放弃了Lisp这个系统。 后来，他接触到Unix这个系统，并且发现，Unix在理论与实际上，都可以在不同的机器间进行移植。虽然 Unix 依旧是专利软件， 但至少 Unix 架构上还是比较开放的！于是他开始转而使用Unix系统。

因为Lisp与Unix是不同的系统，所以，他原本已经撰写完毕的软件是无法在Unix上面运行的！为此， 他就开始将软件移植到Unix上面。并且，为了让软件可以在不同的平台上运行， 因此，史托曼将他发展的软件均撰写成可以移植的型态！也就是他都会将程序的源代码公布出来！

- GNU计划的推展[7]：

1984年，史托曼开始GNU计划， 这个计划的目的是：创建一个自由、开放的Unix操作系统（Free Unix）。 但是创建一个操作系统谈何容易啊！而且在当时的GNU是仅有自己一个人单打独斗的史托曼～ 这实在太麻烦，但又不想放弃这个计划，那可怎么办啊？

聪明的史托曼干脆反其道而行～"既然操作系统太复杂，我就先写可以在Unix上面运行的小程序，这总可以了吧？"在这个想法上， 史托曼开始参考Unix上面现有的软件，并依据这些软件的作用开发出功能相同的软件，且开发期间史托曼绝不看其他软件的源代码， 以避免吃上官司。后来一堆人知道免费的GNU软件，并且实际使用后发现与原有的专利软件也差不了太多，于是便转而使用GNU软件， 于是GNU计划逐渐打开知名度。

虽然GNU计划渐渐打开知名度，但是能见度还是不够。这时史托曼又想：不论是什么软件，都得要进行编译成为二进制文件（binary program）后才能够执行，如果能够写出一个不错的编译器，那不就是大家都需要的软件了吗？ 因此他便开始撰写C语言的编译器，那就是现在

相当有名的GNU C Compiler（gcc）！ 这个点相当的重要！这是因为C语言编译器版本众多，但都是专利软件， 如果他写的C编译器够棒，性能够佳，那么将会大大的让GNU计划出现在众人眼前！如果忘记啥是编译器， 请回到第零章去瞧瞧编译程序吧！

但开始撰写GCC时并不顺利，为此，他先转而将他原先就已经写过的Emacs编辑器写成可以在Unix上面跑的软件，并公布源代码。 Emacs是一种程序编辑器，他可以在使用者撰写程序的过程中就进行程序语法的检验，此一功能可以减少程序设计师除错的时间！ 因为Emacs太优秀了，因此，很多人便直接向他购买。

此时网际网络尚未流行，所以，史托曼便借着Emacs以磁带（tape）出售，赚了一点钱 ，进而开始全力撰写其他软件。并且成立自由软件基金会（FSF, Free Software Foundation），请更多工程师与志工撰写软件。终于还是完成了GCC，这比Emacs还更有帮助！ 此外，他还撰写了更多可以被调用的C函数库（GNU C library），以及可以被使用来操作操作系统的基本接口BASH shell！ 这些都在1990年左右完成了！



Tips 如果纯粹使用文字编辑器来编辑程序的话，那么程序语法如果写错时，只能利用编译时发生的错误讯息来修订了，这样实在很没有效率。 Emacs则是一个很棒的编辑器！注意！是编辑（editor）而非编译（compiler）！ 他可以很快的立刻显示出你写入的语法可能有错误的地方，这对于程序设计师来说， 实在是一个好到不能再好的工具了！所以才会这么的受到欢迎啊！

- GNU的通用公共许可证：

到了1985年，为了避免GNU所开发的自由软件被其他人所利用而成为专利软件， 所以他与律师草拟了有名的通用公共许可证（General Public License, GPL）， 并且称呼他为copyleft（相对于专利软件的copyright！）。 关于GPL的相关内容我们在下一个小节继续谈论，在这里，必须要说明的是， 由于有GNU所开发的几个重要软件，如：

- Emacs
- GNU C （GCC）
- GNU C Library （glibc）
- Bash shell

造成后来很多的软件开发者可以借由这些基础的工具来进行程序开发！ 进一步壮大了自由软件团体！这是很重要的！不过，对于GNU的最初构想 “创建一个自由的Unix操作系统”来说，有这些优秀的程序是仍无法满足， 因为，当下并没有“自由的Unix核心”存在...所以这些软件仍只能在那些有专利的 Unix平台上工作～～一直到Linux的出现...更多的FSF开发的软件可以参考如下网页：

- https://www.fsf.org/resources



Tips 事实上，GNU 自己开发的核心称为 hurd， 是一个架构相当先进的核心。不过由于开发者在开发的过程中对于系统的要求太过于严谨，因此推出的时程一再延后，所以才有后来 Linux 的开发！

- 1988年：图形接口XFree86计划

有鉴于图形使用者接口（Graphical User Interface, GUI） 的需求日益加重，在1984年由MIT与其他协力厂商首次发表了X Window System ，并且更在1988年成立了非营利性质的XFree86这个组织。所谓的XFree86其实是 X Window System + Free + x86的整合名称呢！而这个XFree86的GUI接口更在Linux的核心1.0版于1994年释出时，整合于Linux操作系统当中！



Tips 为什么称图形使用者接口为X呢？因为由英文单字来看，Window的W接的就是X啦！意指Window的下一版就是了！ 需注意的是，X Window并不是X Windows喔！

- 1991年：芬兰大学生Linus Torvalds的一则简讯

到了1991年，芬兰的赫尔辛基大学的Linus Torvalds在BBS上面贴了一则消息， 宣称他以 bash, gcc等 GNU 的工具写了一个小小的核心程序，该核心程序单纯是个玩具，不像 GNU 那么专业。 不过该核心程序可以在Intel的386机器上面运行就是了。这让很多人很感兴趣！从此开始了Linux不平凡的路程！

1.1.3 关于GNU计划、自由软件与开放源代码

GNU计划对于整个自由软件与开放源代码软件来说是占有非常重要的角色！下面我们就来谈谈这咚咚吧！

- 自由软件的活动：

1984年创立GNU计划与FSF基金会的Stallman先生认为，写程序最大的快乐就是让自己发展的良好的软件让大家来使用了！ 另外，如果使用方撰写程序的能力比自己强，那么当对方修改完自己的程序并且回传修改后的程序码给自己，那自己的程序撰写功力无形中就更往上爬了！ 这就是最早之前 AI 实验室的骇客风格！

而既然程序是想要分享给大家使用的，不过，每个人所使用的计算机软硬件并不相同， 既然如此的话，那么该程序的源代码（Source code）就应该要同时释出， 这样才能方便大家修改而适用于每个人的计算机中呢！这个将源代码连同软件程序释出的举动，在 GNU 计划的范畴之内就称为自由软件（Free Software）运动！

此外，史托曼同时认为，如果你将你程序的Source code分享出来时，若该程序是很优秀的，那么将会有很多人使用， 而每个人对于该程序都可以查阅source code，无形之中，就会有一票人帮你除错啰！ 你的这支程序将会越来越壮大！越来越优秀呢！

- 自由软件的版权GNU GPL：

而为了避免自己的开发出来的Open source自由软件被拿去做成专利软件， 于是Stallman同时将GNU与FSF发展出来的软件，都挂上GPL的版权宣告～ 这个FSF的核心观念是“版权制度是促进社会进步的手段， 版权本身不是自然权力。”对于FSF有兴趣或者对于GNU想要更深入的了解时，请参考朝阳科技大学洪朝贵教授的网站 http://people.ofset.org/~ckhung/a/c_83.php，或直接到GNU去： http://www.gnu.org 里面有更为深入的解说！



Tips 为什么要称为GNU呢？其实GNU是GNU's Not Unix的缩写，意思是说，GNU并不是Unix啊！那么GNU又是什么呢？ 就是GNU's Not Unix嘛！.....如果你写过程序就会知道，这个GNU = GNU's Not Unix可是无穷循环啊！忙碌～

另外，什么是Open Source呢？所谓的source code是程序发展者写出的原始程序码， Open Source就是，软件在发布时，同时将作者的源代码一起公布的意思！

- 自由（Free）的真谛：

那么这个GPL（GNU General Public License, GPL）是什么玩意儿？ 为什么要将自由软件挂上GPL的“版权宣告”呢？这个版权宣告对于作者有何好处？ 首先，Stallman对GPL一直是强调Free的，这个Free的意思是这样的：

"Free software" is a matter of liberty, not price. To understand the concept, you should think of "free speech", not "free beer". "Free software" refers to the users' freedom to run, copy, distribute, study, change, and improve the software

大意是说，Free Software（自由软件）是一种自由的权力，并非是“价格！” 举例来说，你可以拥有自由呼吸的权力、你拥有自由发表言论的权力， 但是，这并不代表你可以到处喝“免费的啤酒！（free beer）”，也就是说， 自由软件的重点并不是指“免费”的，而是指具有“自由度, freedom”的软件， 史托曼进一步说明了自由度的意义是： 使用者可以自由的执行、复制、再发行、学习、修改与强化自由软件。

这无疑是个好消息！因为如此一来，你所拿到的软件可能原先只能在Unix上面跑， 但是经过源代码的修改之后，你将可以拿他在Linux或者是Windows上面来跑！总之， 一个软件挂上了GPL版权宣告之后，他自然就成了自由软件！这个软件就具有下面的特色：

- 取得软件与源代码：你可以根据自己的需求来执行这个自由软件；
- 复制：你可以自由的复制该软件；
- 修改：你可以将取得的源代码进行程序修改工作，使之适合你的工作；
- 再发行：你可以将你修改过的程序，再度的自由发行，而不会与原先的撰写者冲突；
- 回馈：你应该将你修改过的程序码回馈于社群！

但请特别留意，你所修改的任何一个自由软件都不应该也不能这样：

- 修改授权：你不能将一个GPL授权的自由软件，在你修改后而将他取消GPL授权～
- 单纯贩卖：你不能单纯的贩卖自由软件。

也就是说，既然GPL是站在互助互利的角度上去开发的，你自然不应该将大家的成果占为己有， 对吧！因此你当然不可以将一个GPL软件的授权取消，即使你已经对该软件进行大幅度的修改！ 那么自由软件也不能贩卖吗？当然不是！还记得上一个小节里面， 我们提到史托曼借由贩卖Emacs取得一些经费，让自己生活不至于匮乏吧？是的！ 自由软件是可以贩售的，不过，不可仅贩售该软件，应同时搭配售后服务与相关手册～ 这些可就需要工本费了呢！

- 自由软件与商业行为：

很多人还是有疑问，目前不是有很多Linux开发商吗？为何他们可以贩售Linux这个GPL授权的软件？ 原因很简单，因为他们大多都是贩售“售后服务！”所以，他们所使用的自由软件， 都可以在他们的网站上面下载！（当然，每个厂商他们自己开发的工具软件就不是GPL的授权软件了！） 但是，你可以购买他们的Linux光盘，如果你购买了光盘，他们会提供相关的手册说明文档， 同时也会提供你数年不等的谘询、售后服务、软件升级与其他协力工作等等的附加价值！

所以说，目前自由软件工作者，他们所赖以维生的，几乎都是在“服务”这个领域呢！ 毕竟自由软件并不是每个人都会撰写，有人有需要你的自由软件时，他就会请求你的协助， 此时，你就可以通过服务来收费了！这样来说， 自由软件确实还是具有商业空间的喔！



Tips 很多人对于GPL授权一直很疑惑，对于GPL的商业行为更是无法接受！ 关于这一点，鸟哥在这里还是要再次的申明，GPL是可以从事商业行为的！ 而很多的作者也是借由这些商业行为来得以取得生活所需，更进一步去发展更优秀的自由软件！ 千万不要听到“商业”就排斥！这对于发展优良软件的朋友来说，是不礼貌的！

上面提到的大多是与使用者有关的项目，那么 GPL 对于自由软件的作者有何优点呢？大致的优点有这些：

- 软件安全性较佳；
- 软件执行性能较佳；
- 软件除错时间较短；
- 贡献的源代码永远都存在。

这是因为既然是提供源代码的自由软件，那么你的程序码将会有很多人帮你查阅， 如此一来，程序的漏洞与程序的优化将会进展的很快！所以，在安全性与性能上面， 自由软件一点都不输给商业软件喔！此外，因为GPL授权当中，修改者并不能修改授权， 因此，你如果曾经贡献过程序码，嘿嘿！你将名留青史呢！不错吧！ ^_^

对于程序开发者来说，GPL实在是一个非常好的授权，因为大家可以互相学习对方的程序撰写技巧， 而且自己写的程序也有人可以帮忙除错。那你会问啊，对于我们这些广大的终端用户，GPL有没有什么好处啊？有啊！当然有！ 虽然终端用户或许不会自己编译程序码或者是帮人家除错，但是终端用户使用的软件绝大部分就是GPL的软件， 全世界有一大票的工程师在帮你维护你的系统，这难道不是一件非常棒的事吗？ ^_^



Tips 就跟人类社会的科技会进步一样，授权也会进步喔！因应源代码分区与重组的问题，与其他开源软件的授权包容性，以及最重要的数码版权管理 （Digital Rights Management, DRM） 等问题，GPL 目前已经出到第三版 GPLv3。但是，目前使用最广泛的，还是 GPLv2 喔！包括 Linux 核心就还是使用 GPLv2 的说！

- 开放源代码：

由于自由软件使用的英文为 free software，这个 free 在英文是有两种以上不同的意义，除了自由之外，免费也是这个单字！ 因为有这些额外的联想，因此许多的商业公司对于投入自由软件方面确实是有些疑虑存在的！许多人对于这个情况总是有些担心～

为了解决这个困扰，1998 年成立的“开放源代码促进会 （Open Source Initiative）”提出了开放源代码 （Open Source，亦可简称开源软件） 这一名词！ 另外，并非软件可以被读取源代码就可以被称为开源软件喔！该软件的授权必须要符合下面的基本需求，才可以算是 open source 的软件哩！[8]

- 公布源代码且用户具有修改权：用户可以任意的修改与编译程序码，这点与自由软件差异不大；
- 任意的再散佈：该程序码全部或部份可以被贩售，且程序码可成为其他软件的元件之一，作者不该宣称具有拥有权或收取其他额外费用。

- 必须允许修改或衍生的作品，且可让再发布的软件使用相似的授权来发表即可。
- 承上，用户可使用与原本软件不同的名称或编号来散佈。
- 不可限制某些个人或团体的使用权
- 不可限制某些领域的应用：例如不可限制不能用于商业行为或者是学术行为等特殊领域等等
- 不可限制在某些产品当中，亦即程序码可以应用于多种不同产品中。
- 不可具有排他条款，例如不可限制本程序码不能用于教育类的研究中，诸如此类。

根据上面的定义，GPL 自由软件也可以算是开源软件的一个，只是对于商业应用的限止稍微多一些而已。 与 GPL 自由软件相比，其他开源软件的授权可能比较轻松喔！比较轻松的部份包括：再发布的授权可以跟原本的软件不同； 另外，开源软件的全部或部份可作为其他软件的一部分，且其他软件无须使用与开源软件相同的授权来发布！这跟GPL自由软件差异就大了！ 自由软件的GPL授权规定，任何软件只要用了GPL的全部或部份程序码，那么该软件就得要使用GPL的授权！这对于自由软件的保障相当大！ 但对于想要保有商业公司自己的商业机密的专属软件来说，要使用GPL授权还是怕怕的！这也是后来商业公司拥抱其他 open source 开源软件授权的缘故！因为可以用于商业行为啰！更多的差异或许可以参考一下开源促进会的说明。

另外，Open source 这个名词只是一个指引，而实际上并不是先有 open source 才有相关的授权。早在 open source 出来之前就有些开源软件的授权存在了 （例如 GPL 啊！）！ 不过有 open source 这个名词之后，大家才更了解到开源软件授权的意义就是了。那常见的开放源代码授权有哪些呢？

- Apache License 2.0
- BSD 3-Clause "New" or "Revised" license
- BSD 2-Clause "Simplified" or "FreeBSD" license
- GNU General Public License （GPL）
- GNU Library or "Lesser" General Public License （LGPL）
- MIT license
- Mozilla Public License 2.0
- Common Development and Distribution License

鸟哥也不是软件授权的高手！每个授权详细的内容也可以参考 OSI 协会的介绍啦[9]。



Tips 如前所述，GPL 也是合乎 Open source 所定义的授权之一，只是它更着重于保护自由软件本身的学习与发展就是了！那如果你想要开发开源软件时， 到底使用哪种授权比较好呢？其实跟你对这个软件的未来走向的定义有关啦！简单的来说，如果你的软件未来你允许它用

于商业活动中， 可以考虑 BSD 之类的授权，如果你的软件希望少一些商业色彩，GPLv2 大概是不二选择啰！那如果你的软件允许分支开发， 甚至可以考虑分成两种版本分别授权哩！ ^_^

- 专属软件/专利软件 （close source）

相对于Open Source的软件会释出源代码，Close source的程序则仅推出可执行的二进制程序（binary program）而已。 这种软件的优点是有专人维护，你不需要去更动他；缺点则是灵活度大打折扣，使用者无法变更该程序成为自己想要的样式！ 此外，若有木马程序或者安全漏洞，将会花上相当长的一段时间来除错！这也是所谓专利软件（copyright）常见的软件出售方式。

虽然专利软件常常代表就是需要花钱去购买，不过有些专利软件还是可以“免费”提供福斯使用的！免费的专利软件代表的授权模式有：

- Freeware： http://en.wikipedia.org/wiki/Freeware 不同于Free software，Freeware为“免费软件”而非“自由软件！”虽然它是免费的软件，但是不见得要公布其源代码， 端看释出者的意见啰！这个东西与Open Source毕竟是不太相同的东西喔！此外，目前很多标榜免费软件的程序很多都有小问题！ 例如假藉免费软件的名义，实施使用者数据窃取的目的！ 所以“来路不明的软件请勿安装！”

- Shareware： http://en.wikipedia.org/wiki/Shareware 共享软件这个名词就有趣了！与免费软件有点类似的是，Shareware在使用初期，它也是免费的，但是， 到了所谓的“试用期限”之后，你就必须要选择“付费后继续使用”或者“将它移除”的宿命～ 通常，这些共享软件都会自行撰写失效程序，让你在试用期限之后就无法使用该软件。

1.2 Torvalds的Linux发展

我们前面一节当中，提到了Unix的历史，也提到了Linux是由Torvalds这个芬兰人所发明的。那么为何托瓦兹可以发明Linux呢？ 凭空想像而来的？还是有什么渊源？这里我们就来谈一谈啰！

1.2.1 与Minix之间

Linus Torvalds（托瓦兹, 1969年出生, [10]）的外祖父是赫尔辛基大学的统计学家， 他的外祖父为了让自己的小孙子能够学点东西，所以从小就将托瓦兹带到身边来管理一些微计算机。在这个时期，托瓦兹接触了组合语言（Assembly Language），那是一种直接与芯片对谈的程序语言，也就是所谓的低阶语言。 必须要很了解硬件的架构，否则很难以组合语言撰写程序的。

在1988年间，托瓦兹顺利的进入了赫尔辛基大学，并选读了计算机科学系。在就学期间，因为学业的需要与自己的兴趣， 托瓦兹接触到了Unix这个操作系统。当时整个赫尔辛基只有一部最新的Unix系统，同时仅提供16个终端机（terminal）。 还记得我们上一节刚刚提过的，早期的计算机仅有主机具有运算功能，terminal仅负责提供Input/Output而已。在这种情况下， 实在很难满足托瓦兹的需求，因为.....光是等待使用Unix的时间，就很耗时～为此，他不禁想到：“我何不自己搞一部Unix来玩？”不过，就如同Stallman当初的GNU计划一样，要写核心程序，谈何容易～

不过，幸运之神并未背离托瓦兹，因为不久之后，他就知道有一个类似Unix的系统， 并且与Unix完全相容，还可以在Intel 386机器上面跑的操作系统， 那就是我们上一节提过的，谭宁邦教授为了教育需要而撰写的Minix系统！ 他在购买了最新的Intel 386的个人计算机后，就立即安装了Minix这个操作系统。 另外，上个小节当中也谈到，Minix这个操作系统是有附上源代码的， 所以托瓦兹也经由这个源代码学习到了很多的核心程序设计的设计概念喔！

1.2.2 对386硬件的多任务测试

事实上，托瓦兹对于个人计算机的CPU其实并不满意，因为他之前碰的计算机都是工作站型的计算机， 这类计算机的CPU特色就是可以进行“多任务处理”的能力。什么是多任务呢？理论上， 一个CPU在一个时间内仅能进行一个程序， 那如果有两个以上的程序同时出现到系统中呢？举例来说， 你可以在现今的计算机中同时打开两个以上的办公软件，例如电子试算表与文书处理软件。 这个同时打开的动作代表着这两个程序同时要交给CPU来处理～

啊！CPU一个时间点内仅能处理一个程序，那怎么办？没关系，这个时候如果具有多任务能力的CPU就会在不同的程序间切换～ 还记得前一章谈到的CPU频率吧？假设CPU频率为1GHz的话，那表示CPU一秒钟可以进行109次工作。 假设CPU对每个程序都只进行1000次

运行周期，然后就得要切换到下个程序的话，那么CPU一秒钟就能够切换106次呢！（当然啦，切换工作这件事情也会花去一些CPU时间，不过这里暂不讨论）。这么快的处理速度下，你会发现， 两个程序感觉上几乎是同步在进行啦！



Tips 为什么有的时候我同时开两个文件（假设为A, B文件）所花的时间， 要比开完A再去开B文件的时间还要多？现在是否稍微可以理解？ 因为如果同时打开的话，CPU就必须要在两个工作之间不停的切换～ 而切换的动作还是会耗去一些CPU时间的！所以啰，同时启用两个以上的工作在一个CPU上， 要比一个一个的执行还要耗时一点。这也是为何现在CPU开发商要整合多个CPU于一个芯片中！ 也是为何在运行情况比较复杂的服务器上，需要比较多的CPU负责的原因！

早期Intel x86架构计算机不是很受重视的原因，就是因为x86的芯片对于多任务的处理不佳，CPU在不同的工作之间切换不是很顺畅。但是这个情况在386计算机推出后，有很大的改善。托瓦兹在得知新的386芯片的相关信息后，他认为，以性能价格比的观点来看， Intel的386相当的便宜，所以在性能上也就稍微可以将就将就 ^_^。最终他就贷款去买了一部Intel的386来玩。

早期的计算机性能没有现在这么好，所以压榨计算机性能就成了工程师的一项癖好！ 托瓦兹本人早期是玩组合语言的，组合语言对于硬件有很密切的关系，托瓦兹自己也说："我始终是个性能癖"^^。 为了彻底发挥*386*的性能，于是托瓦兹花了不少时间在测试*386*机器上！ 他的重要测试就是在测试*386*的多功性能。首先，他写了三个小程序，一个程序会持续输出*A*、一个会持续输出*B*， 最后一个会将两个程序进行切换。他将三个程序同时执行，结果，他看到屏幕上很顺利的一直出现*ABABAB......* 他知道，他成功了！ ^^

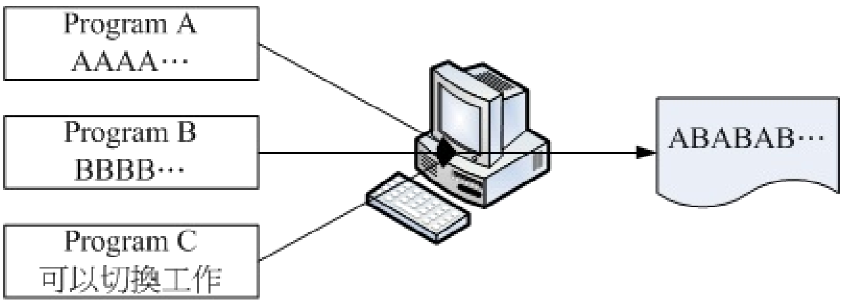


图1.2.1、386计算机的多任务测试



Tips 要达到多工（multitasking）的环境，除了硬件（主要是CPU）需要能够具有多任务的特性外，操作系统也需要支持这个功能喔！ 一些不具有多任务特性的操作系统，想要同时执行两个程序是不可能的。除非先被执行的程序执行完毕，否则， 后面的程序不可能被主动执行。

至于多任务的操作系统中，每个程序被执行时，都会有一个最大CPU使用时间，若该工作运行的时间超过这个CPU使用时间时， 该工作就会先被丢出CPU的运行中，而再度的进入核心工作调度中等待下一次被CPU取用来运行。

这有点像在开记者会啦，主持人（CPU）会问“谁要发问”？一群记者（工作程序） 就会举手（看谁的工作重要！），先举手的自然就被允许发问，问完之后， 主持人又会问一次谁要发问，当然，所有人（包括刚刚那个记者）都可以举手！ 如此一次一次的将工作给他完成啊！ ^_^ 多任务的环境对于复杂的工作情况，帮助很大喔！

1.2.3 初次释出Linux 0.02

探索完386的硬件性能之后，终于拿到Minix并且安装在托瓦兹的386计算机上之后，托瓦兹跟BBS上面一堆工程师一样， 他发现Minix虽然真的很棒，但是谭宁邦教授就是不愿意进行功能的加强，导致一堆工程师在操作系统功能上面的欲求不满！ 这个时候年轻的托瓦兹就想：“既然如此，那我何不自己来改写一个我想要的操作系统？” 于是他就开始了核心程序的撰写了。

撰写程序需要什么呢？首先需要的是能够进行工作的环境，再来则是可以将源代码编译成为可可执行文件的编译器。 好在有GNU计划提供的bash工作环境软件以及gcc编译器等自由软件， 让托瓦兹得以顺利的撰写核心程序。他参考Minix的设计理念与书上的程序码，然后仔细研究出386个人计算机的性能最优化， 然后使用GNU的自由软件将核心程序码与386紧紧的结合在一起，最终写出他所需要的核心程序。 而这个小玩意竟然真的可以在386上面顺利的跑起来～还可以读取Minix的文件系统。 真是太好了！不过还不够，他希望这个程序可以获得大家的一些修改建议， 于是他便将这个核心放置在网络上提供大家下载，同时在BBS上面贴了一则消息：

```
Hello everybody out there using minix-
I'm doing a （free） operation system （just a hobby,
won't be big and professional like gnu） for 386（486） AT clones.

I've currently ported bash （1.08） and gcc （1.40），
and things seem to work. This implies that i'll get
something practical within a few months, and I'd like to know
what features most people want. Any suggestions are welcome,
but I won't promise I'll implement them :-）
```

他说，他完成了一个小小的操作系统，这个核心是用在386机器上的， 同时，他真的仅是好玩，并不是想要做一个跟GNU一样大的计划！ 另外，他希望能够得到更多人的建议与回馈来发展这个操作系统！这个概念跟Minix刚好背道而驰呢！ 这则新闻引起很多人的注意，他们也

去托瓦兹提供的网站上下载了这个核心来安装。 有趣的是，因为托瓦兹放置核心的那个FTP网站的目录为：Linux， 从此，大家便称这个核心为Linux了。（请注意，此时的Linux就是那个kernel喔！ 另外，托瓦兹所丢到该目录下的第一个核心版本为0.02呢！）

同时，为了让自己的Linux能够相容于Unix系统，于是托瓦兹开始将一些能够在Unix上面运行的软件拿来在Linux上面跑。 不过，他发现到有很多的软件无法在Linux这个核心上运行。这个时候他有两种作法， 一种是修改软件，让该软件可以在Linux上跑， 另一种则是修改Linux，让Linux符合软件能够运行的规范！ 由于Linux希望能够相容于Unix，于是托瓦兹选择了第二个作法“修改Linux”！ 为了让所有的软件都可以在Linux上执行，于是托瓦兹开始参考标准的POSIX规范。



Tips POSIX是可携式操作系统接口（Portable Operating System Interface）的缩写，重点在规范核心与应用程序之间的接口， 这是由美国电器与电子工程师学会（IEEE）所发布的一项标准喔！

这个正确的决定让Linux在起步的时候体质就比别人优良～因为POSIX标准主要是针对Unix与一些软件运行时候的标准规范， 只要依据这些标准规范来设计的核心与软件，理论上，就可以搭配在一起执行了。 而Linux的发展就是依据这个POSIX的标准规范，Unix上面的软件也是遵循这个规范来设计的， 如此一来，让Linux很容易就与Unix相容共享互有的软件了！同时，因为Linux直接放置在网络下，提供大家下载， 所以在流通的速度上相当的快！导致Linux的使用率大增！这些都是造成Linux大受欢迎的几个重要因素呢！



Tips 其实托瓦兹有意无意之间常常会透露他自己是个只喜欢玩 （Just for Fun） 的怪人！ Linux 一开始也只是托瓦兹的一个作业发展出来的玩具而已。 他也说，如果 Minix 或 hurd 这两个中的任何一个系统可以提早开发出他想要的功能与环境，也许他根本不会想要自己开发一个 Linux 哩！ 哇！人类智慧眞是没有极限！各位啊： 1）要先有基础知识与技能、2）有了第一点后，要勇于挑战权威、3）把你们的玩具发扬光大吧！ ^_^

1.2.4 Linux 的发展：虚拟团队的产生

Linux能够成功除了托瓦兹个人的理念与力量之外，其实还有个最重要的团队！

- 单一个人维护阶段

Linux虽然是托瓦兹发明的，而且内容还绝不会涉及专利软件的版权问题。不过，如果单靠托瓦兹自己一个人的话，那么Linux要茁壮实在很困难～ 因为一个人的力量是很有限的。好在托瓦兹选择Linux的开发方式相当的务实！首先，他将释出的Linux核心放置在FTP上面，并请告知大家新的版本信息， 等到使用者下载了这个核心并且安装之后，如果发生问题，或者是由于特殊需求亟需某些硬件的驱动程序，那么这些使用者就会主动回报给托瓦兹。 在托瓦兹能够解决的问题范围内，他都能很快速的进行Linux核心的更新与除错。

- 广大骇客志工加入阶段

不过，托瓦兹总是有些硬件无法取得的啊，那么他当然无法帮助进行驱动程序的撰写与相关软件的改良。 这个时候，就会有些志工跳出来说："这个硬件我有，我来帮忙写相关的驱动程序。" 因为Linux的核心是Open Source的，骇客志工们很容易就能够跟随Linux的原本设计架构， 并且写出相容的驱动程序或者软件。志工们写完的驱动程序与软件托瓦兹是如何看待的呢？ 首先，他将该驱动程序/软件带入核心中，并且加以测试。 只要测试可以运行，并且没有什么主要的大问题，那么他就会很乐意的将志工们写的程序码加入核心中！

总之，托瓦兹是个很务实的人，对于Linux核心所欠缺的项目，他总是"先求有且能跑， 再求进一步改良"的心态！这让Linux使用者与志工得到相当大的鼓励！ 因为Linux的进步太快了！使用者要求虚拟内存，结果不到一个星期推出的新版Linux就有了！ 这不得不让人佩服啊！

另外，为因应这种随时都有程序码加入的状况，于是Linux便逐渐发展成具有模块的功能！ 亦即是将某些功能独立出于核心外，在需要的时候才载入到核心中。如此一来， 如果有新的硬件驱动程序或者其他协定的程序码进来时，就可以模块化， 大大的增加了Linux核心的可维护能力！



Tips 核心是一组程序，如果这组程序每次加入新的功能都得要重新编译与改版的话会变成如何？ 想像一下，如果你只是换了显卡就得要重新安装新的Windows操作系统，会不会傻眼？模块化之后，原本的核心程序不需要更动，你可以直接将他想成是"驱动程序"即可！ ^_^

- 核心功能细部分工发展阶段

后来，因为Linux核心加入了太多的功能，光靠托瓦兹一个人进行核心的实际测试并加入核心原始程序实在太费力～ 结果，就有很多的朋友跳出来帮忙这个前置作业！例如考克斯（Alan Cox）、与崔迪（Stephen Tweedie）等等， 这些重要的副手会先将来自志工们的修补程序或者新功能的程序码进行测试， 并且结果上传给托瓦兹看，让托瓦兹作最后核心加入的源代码的选择与整并！ 这个分层负责的结果，让Linux的发展更加的容易！

特别值得注意的是，这些托瓦兹的Linux发展副手，以及自愿传送修补程序的骇客志工， 其实都没有见过面，而且彼此在地球的各个角落，大家群策群力的共同发展出现今的Linux， 我们称这群人为虚拟团队！而为了虚拟团队数据的传输，于是Linux便成立的核心网站：http://www.kernel.org！

而这群素未谋面的虚拟团队们，在1994年终于完成的Linux的核心正式版！version 1.0。 这一版同时还加入了X Window System的支持呢！且于1996年完成了2.0版、2011 年释出 3.0 版，更于 2015 年 4 月释出了 4.0 版哩！ 发展相当迅速喔！此外，托瓦兹指明了企鹅为Linux的吉祥物。



Tips 奇怪的是，托瓦兹是因为小时候去动物园被企鹅咬了一口念念不忘， 而正式的2.0推出时，大家要他想一个吉祥物。他在想也想不到什么动物的情况下， 就将这个念念不忘的企鹅当成了Linux的吉祥物了......

Linux由于托瓦兹是针对386写的，跟386硬件的相关性很强，所以， 早期的Linux确实是不具有移植性的。不过，大家知道Open source的好处就是， 可以修改程序码去适合作业的环境。因此，在1994年以后，Linux便被开发到很多的硬件上面去了！ 目前除了x86之外，IBM、HP等等公司出的硬件也都有被Linux所支持呢！甚至于小型单板计算机 （树莓派/香蕉派等） 与手持设备 （智能手机、平板电脑） 的 ARM 架构系统，大多也是使用 Linux 核心喔！

1.2.5 Linux的核心版本

Linux的核心版本编号有点类似如下的样子：

```
3.10.0-123.el7.x86_64
主版本.次版本.释出版本-修改版本
```

虽然编号就是如上的方式来编的，不过依据 Linux 核心的发展期程，核心版本的定义有点不太相同喔！

- 奇数、偶数版本分类

在 2.6.x 版本以前，托瓦兹将核心的发展趋势分为两股，并根据这两股核心的发展分别给予不同的核心编号，那就是：

- 主、次版本为奇数：发展中版本（development） 如2.5.xx，这种核心版本主要用在测试与发展新功能，所以通常这种版本仅有核心开发工程师会使用。 如果有新增的核心程序码，会加到这种版本当中，等到众多工程师测试没问题后，才加入下一版的稳定核心

中；

- 主、次版本为偶数：稳定版本（stable） 如2.6.xx，等到核心功能发展成熟后会加到这类的版本中，主要用在一般家用计算机以及企业版本中。 重点在于提供使用者一个相对稳定的Linux作业环境平台。

至于释出版本则是在主、次版本架构不变的情况下，新增的功能累积到一定的程度后所新释出的核心版本。 而由于Linux核心是使用GPL的授权，因此大家都能够进行核心程序码的修改。因此，如果你有针对某个版本的核心修改过部分的程序码， 那么那个被修改过的新的核心版本就可以加上所谓的修改版本了。

- 主线版本、长期维护版本（longterm version）

不过，这种奇数、偶数的编号格式在 3.0 推出之后就失效了。从 3.0 版开始，核心主要依据主线版本 （MainLine） 来开发，开发完毕后会往下一个主线版本进行。 例如 3.10 就是在 3.9 的架构下继续开发出来的新的主线版本。通常新一版的主线版本大约在 2~3 个月会被提出喔！ 之所以会有新的主线版本，是因为有加入新功能之故。现在 （2015/04） 最新的主线版本已经来到 4.0 版了喔！好快！

而旧的版本在新的主线版本出现之后，会有两种机制来处理，一种机制为结束开发 （End of Live, EOL），亦即该程序码已经结束，不会有继续维护的状态。 另外一种机制为保持该版本的持续维护，亦即为长期维护版本 （Longterm）！例如 3.10 即为一个长期维护版本，这个版本的程序码会被持续维护，若程序码有 bug 或其他问题， 核心维护者会持续进行程序码的更新维护喔！

所以啰，如果你想要使用 Linux 核心来开发你的系统，那么当然要选择长期支持的版本才行！要判断你的 Linux 核心是否为长期支持的版本， 可以使用“ uname -r ”来查阅核心版本，然后对照下列链接来了解其对应值喔！

- https://www.kernel.org/releases.html

- Linux 核心版本与 Linux 发布商版本

Linux核心版本与distribution （下个小节会谈到） 的版本并不相同，很多朋友常常上网问到：“我的Linux是7.x版，请问....”之类的留言， 这是不对的提问方式，因为所谓的Linux版本指的应该是核心版本， 而目前最新的核心版本应该是4.0.0（2015/04） 才对，并不会有7.x的版本出现的。

你常用的Linux系统则应该说明为distribution才对！因此，如果以CentOS这个distribution来说， 你应该说：“我用的Linux是CentOS这个 distribution，版本为7.x 版，请问....”才对喔！



Tips 当你有任何问题想要在Linux论坛发言时，请务必仔细的说明你的distribution版本， 因为虽然各家distributions使用的都是Linux核心，不过每家distributions所选用的软件以及他们自己发展的工具并不相同， 多少还是有点差异，所以留言时得要先声明distribution的版本才行喔！ ^_^

1.2.6 Linux distributions

好了，经过上面的说明，我们知道了Linux其实就是一个操作系统最底层的核心及其提供的核心工具。 他是GNU GPL授权模式，所以，任何人均可取得源代码与可执行这个核心程序，并且可以修改。 此外，因为Linux参考POSIX设计规范，于是相容于Unix操作系统，故亦可称之为Unix Like的一种。



Tips 鸟哥曾在上课的时候问过同学："什么是Unix Like啊"？可爱的同学们回答的答案是："就是很喜欢（like）Unix啦！" 囧rz...那个like是"很像"啦！所以Unix like是"很像Unix的操作系统"哩！

- 可完整安装的Linux发布套件

Linux的出现让GNU计划放下了心里的一块大石头，因为GNU一直以来就是缺乏了核心程序，导致他们的GNU自由软件只能在其他的Unix上面跑。既然目前有Linux出现了，且 Linux也用了很多的GNU相关软件，所以Stallman认为Linux的全名应该称之为GNU/Linux呢！ 不管怎么说，Linux实在很不错，让GNU软件大多以Linux为主要操作系统来进行开发， 此外，很多其他的自由软件团队，例如postfix, vsftpd, apache等等也都有以Linux 为开发测试平台的计划出现！如此一来，Linux除了主要的核心程序外，可以在Linux 上面运行的软件也越来越多，如果有心，就能够将一个完整的Linux操作系统搞定了！

虽然由Torvalds负责开发的Linux仅具有Kernel与Kernel提供的工具， 不过，如上所述，很多的软件已经可以在Linux上面运行了，因此，"Linux + 各种软件"就可以完成一个相当完整的操作系统了。 不过，要完成这样的操作系统......还真难～ 因为Linux早期都是由骇客工程师所开发维护的，他们并没有考虑到一般使用者的能力......

为了让使用者能够接触到Linux，于是很多的商业公司或非营利团体， 就将Linux Kernel（含tools）与可运行的软件整合起来，加上自己具有创意的工具程序， 这个工具程序可以让使用者以光盘/DVD或者通过网络直接安装/管理Linux系统。 这个"Kernel + Softwares + Tools + 可完整安装程序"的咚咚，我们称之为Linux distribution， 一般中文翻译成可完整安装套件，或者Linux发布商套件等。

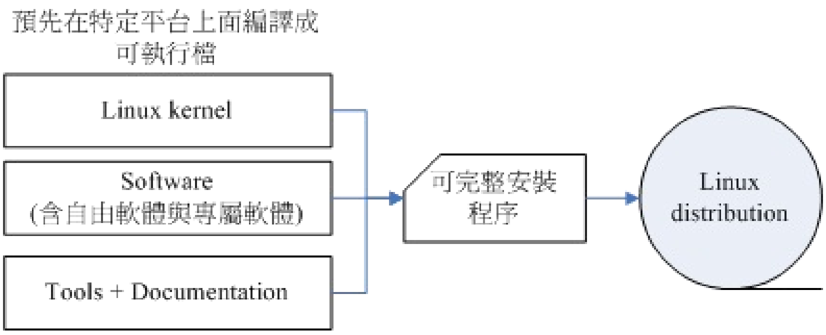


图1.2.2、Linux可完整安装发布套件



Tips 由于Linux核心是由骇客工程师写的，要由源代码安装到x86计算机上面成为可以执行的binary文件， 这个过程可不是人人都会的～所以早期确实只有工程师对Linux有兴趣。 一直到一些社群与商业公司将Linux核心配合自由软件，并提供完整的安装程序， 且制成光盘/DVD后，对于一般使用者来说，Linux才越来越具有吸引力！ 因为只要一直“下一步”就可以将Linux安装完成啊！ ^_^

由于GNU的GPL授权并非不能从事商业行为，于是很多商业公司便成立来贩售Linux distribution。 而由于Linux的GPL版权宣告，因此，商业公司所贩售的Linux distributions通常也都可以从Internet上面来下载的！ 此外，如果你想要其他商业公司的服务，那么直接向该公司购买光盘来安装，也是一个很不错的方式的！

- 各大Linux Distributions的主要异同：支持标准！

不过，由于发展Linux distributions的社群与公司实在太多了，例如在台湾有名的Red Hat, SuSE, Ubuntu, Fedora, Debian等等，所以很多人都很担心，如此一来每个distribution是否都不相同呢？ 这就不需要担心了，因为每个Linux distributions使用的kernel都是http://www.kernel.org所释出的，而他们所选择的软件，几乎都是目前很知名的软件，重复性相当的高， 例如网页服务器的Apache，电子邮件服务器的Postfix/sendmail，文件服务器的Samba等等。

此外，为了让所有的Linux distributions开发不致于差异太大，且让这些开发商在开发的时候有所依据，还有Linux Standard Base （LSB）等标准来规范开发者，以及目录架构的File system Hierarchy Standard （FHS）标准规范！ 唯一差别的，可能就是该开发者自家所开发出来的管理工具，以及套件管理的模式吧！ 所以说，基本上，每个Linux distributions除了架构的严谨度与选择的套件内容外， 其实差异并不太大啦！ ^_^ 。大家可以选择自己喜好的distribution来安装即可！

- FHS: http://www.pathname.com/fhs/
- LSB: http://www.linuxbase.org/

事实上鸟哥认为distributions主要分为两大系统，一种是使用RPM方式安装软件的系统，包括Red Hat, Fedora, SuSE等都是这类； 一种则是使用Debian的dpkg方式安装软件的系统，包括Debian, Ubuntu, B2D等等。若是加上商业公司或社群单位的分类，那么我们可以简单的用下表来做个解释喔！

	RPM 软件管理	**DPKG** 软件管理	其他未分类
商业公司	RHEL （Red Hat 公司） SuSE （Micro Focus）	Ubuntu （Canonical Ltd.）	
社群单位	Fedora CentOS OpenSuSE	Debian B2D	Gentoo

下面列出几个主要的Linux distributions发行者网址：

- Red Hat: http://www.redhat.com
- SuSE: https://www.suse.com
- Fedora: https://getfedora.org/
- CentOS: http://www.centos.org/
- Debian: http://www.debian.org/
- Ubuntu: http://www.ubuntu.com/
- Gentoo: http://www.gentoo.org/



Tips 到底是要买商业版还是社群版的Linux distribution呢？如果是要装在个人计算机上面做为桌面电脑用的，建议使用社群版， 包括Fedora, Ubuntu, OpenSuSE等等。如果是用在服务器上面的，建议使用商业版本，包括Red Hat, SuSE等。 这是因为社群版通常开发者会加入最新的软件，这些软件可能会有一些bug存在。至于商业版则是经过一段时间的磨合后， 才将稳定的软件放进去。

举例来说，Fedora兜出来的软件套件经过一段时间的维护后，等到该软件稳定到不容易发生错误后， Red Hat才将该软件放到他们最新的释出版本中。所以，Fedora的软件比较经常改版，Red Hat的软件就较少更版。

- Linux在台湾

当然发行套件者不仅于此。但是值得大书特书的，是中文Linux的延伸计划：CLE这个套件！早期的Linux因为是工程师发展的，而这些工程师大多以英文语系的国家为主， 所以Linux对于国人的学习是比较困扰一点。后来由国人发起的CLE计划， 开发很多的中文套件及翻译了很多的英文文件，使得我们目前得以使用中文的Linux呢！ 另外，目前正在开发中的还有台南县卧龙小三等老师们发起的众多自由软件计划， 真是造福很多的朋友啊！

- 自由软件技术交流网： http://freesf.tw/
- B2D: http://b2d-linux.com/

此外，如果只想看看Linux的话，还可以选择所谓的可光盘开机进入Linux的Live CD版本， 亦即是KNOPPIX这个Linux distributions呢！台湾也有阿里巴巴兄维护的中文Live CD喔！

- http://www.knoppix.net/
- 洪老师解释 KNOPPIX: http://people.ofset.org/~ckhung/b/sa/knoppix.php



Tips 对于没有额外的硬盘或者是没有额外的主机的朋友来说，KNOPPIX这个可以利用光盘开机而进入Linux操作系统的Live CD 真的是一个不错的选择！你只要下载了KNOPPIX的镜像文件，然后将他烧录成为CD， 放入你主机的光驱，并在BIOS内设置光盘为第一个开机选项，就可以使用Linux系统了呢！

如果你还想要知道更多的Linux distributions的下载与使用信息，可以参考：

- http://distrowatch.com/

- 选择适合你的Linux distribution

那我到底应该要选择哪一个distributions？就如同我们上面提到的，其实每个distributions差异性并不大！ 不过，由于套件管理的方式主要分为Debian的dpkg及Red Hat系统的RPM方式，目前鸟哥的建议是，先学习以RPM套件管理为主的RHEL/Fedora/SuSE/CentOS等台湾使用者较多的版本， 这样一来，发生问题时，可以提供解决的管道比较多。如果你已经接触过Linux了， 还想要探讨更严谨的Linux版本，那可以考虑使用Debian，如果你是以性能至上来考虑， 那么或许Gentoo是不错的建议！

总之，版本很多，但是各版本差异其实不大，建议你一定要先选定一个版本后， 先彻头彻尾的了解他，那再继续玩其他的版本时，就可以很快的进入状况。 鸟哥的网站仅提供一个版本，不过是以比较基础的方式来介绍的， 因此，如果能够熟练俺这个网站的话，呵呵！哪一个distributions对你来说，都不成问题啦！

不过，如果依据计算机主机的用途来分的话，在台湾鸟哥会这样建议：

- 用于企业环境：建议使用商业版本，例如Red Hat的RHEL或者是SuSE都是很不错的选择！ 毕竟企业的环境强调的是永续的经营，你可不希望网管人员走了之后整个机房的主机都没有人管理吧！ 由于商业版本都会提供客户服务，所以可以降低企业的风险喔！

- 用于个人或教学的服务器环境：要是你的服务器所在环境如果死机还不会造成太大的问题的话， 加上你的环境是在教学的场合当中时（就是说，唔！经费不足的环境啦！）那么可以使用“号称”完全相容商业版RHEL的CentOS。 因为CentOS是抓RHEL的源代码来

重新兜起来的一个Linux distribution，所以号称相容于RHEL。 这一版的软件完全与RHEL相同，在改版的幅度较小，适合于服务器系统的环境；

- 用于个人的桌面电脑：想要尝鲜吗？建议使用很炫的Fedora/Ubuntu等Desktop（桌面环境）使用的版本！ 如果不想要安装Linux的话，那么Fedora或CentOS也有推出Live CD了！也很容易学习喔！

1.3 Linux当前应用的角色

了解了什么是 Linux 之后，再来谈谈，那目前 Linux 用在哪里呢？由于Linux kernel实在是非常的小巧精致，可以在很多强调省电以及较低硬件资源的环境下面执行； 此外，由于Linux distributions整合了非常多非常棒的软件（不论是专利软件或自由软件）， 因此也相当适合目前个人计算机的使用呢！传统上，Linux常见的应用可约略分为企业应用与个人应用两方面， 但这几年很流行的云端运算机制中， 让 Linux 似乎又更有着力点啰！

1.3.1 企业环境的利用

企业对于数码化的目标在于提供消费者或员工一些产品方面的信息 （例如网页介绍），以及整合整个企业内部的数据统一性 （例如统一的帐号管理/文件管理系统等）。另外，某些企业例如金融业等，则强调在数据库、安全强化等重大关键应用。 学术单位则很需要强大的运算能力等。所以企业环境运用Linux作些什么呢？

- 网络服务器：

这是Linux当前最热门的应用了！承袭了Unix高稳定性的良好传统，Linux上面的网络功能特别的稳定与强大！ 此外，由于GNU计划与Linux的GPL授权模式，让很多优秀的软件都在Linux上面发展， 且这些在Linux上面的服务器软件几乎都是自由软件！因此，做为一部网络服务器，例如WWW, Mail Server, File Server等等，Linux绝对是上上之选！当然，这也是Linux的强项！由于 Linux server 的需求强烈，因此许多硬件厂商推出产品时， 还得要特别说明有支持的 Linux distributions 呢！方便提供企业采购部门的规划喔！例如下面的链接可以瞧瞧：

- Dell 公司的 Server 对 OS 的支持度：http://www.dell.com/support/contents/tw/en/twbsd1/article/Product-Support/ Self-support-Knowledgebase/enterprise-resource-center/server-operating-system-support
- HP 公司的支持： http://www8.hp.com/us/en/business-services/it-services.html?compURI=1078888#tab=TAB1
- IBM 公司的支持： http://www-03.ibm.com/systems/hardware/browse/linux/
- VMWare 的虚拟化支持：https://www.vmware.com/support/ws55/doc/intro_supguest_ws.html

从上面的几个大厂的 Linux 支持情况来看，目前 （2015） 支持度比较广泛的依旧是 Red Hat 以及 SuSU 两个大厂喔！ 提估给企业采购的时候参考参考！



Tips 前一阵子参加一个座谈会，会上许多企业界的前辈们在聊，如果想要选择某个 Linux distribution 时，哪个 distribution 会是企业采购时的最爱呢？ 与会的朋友说，要采购吗？看看服务器大厂对于该 distribution 的支持度就知道了！答案是什么？就是上面许多链接的结果啰！ ^_^

- 关键任务的应用（金融数据库、大型企业网管环境）：

由于个人计算机的性能大幅提升且价格便宜，所以金融业与大型企业的环境为了要精实自己机房的机器设备， 因此很多企业渐渐的走向Intel相容的x86主机环境。而这些企业所使用的软件大多使用Unix操作系统平台的软件， 总不能连过去发展的软件都一口气全部换掉吧！所以啰，这个时候符合Unix操作系统标准并且可以在x86上运行的Linux就渐渐崭露头角了！^_^

目前很多金融业界都已经使用Linux做为他们的关键任务应用。所谓的关键任务就是该企业最重要的业务啦！ 举例来说，金融业最重要的就是那些投资者、帐户的数据了，这些数据大多使用数据库系统来作为存取接口， 这些数据很重要吧！很多金融业将这么重要的任务交给了Linux了！你说Linux厉不厉害啊？

- 学术机构的高性能运算任务：

学术机构的研究常常需要自行开发软件，所以对于可作为开发环境的操作系统需求非常的迫切！举例来说， 非常多技职体系的科技大学就很需要这方面的环境，好进行一些毕业专题的制作呢！ 又例如工程界流体力学的数值模式运算、娱乐事业的特效功能处理、软件开发者的工作平台等等。 由于Linux的创造者本身就是个计算机性能癖，所以Linux有强大的运算能力；并且Linux具有支持度相当广泛的GCC编译软件， 因此Linux在这方面的优势可是相当明显的！

举个鸟哥自己的案例好了，鸟哥之前待的研究室有跑一套空气品质模式的数值仿真软件。 这套软件原本只能在Sun的SPARC机器上面跑。后来该软件转向Linux操作系统平台发展， 鸟哥也将自己实验室的数值模式程序由Sun的Solaris平台移植到Linux上面呢！ 据美国环保署内部人员的测试，发现Linux平台的整体硬件费用不但比较便宜（x86系统嘛！）而且速度还比较快呢！

另外，为了加强整体系统的性能，从集计算机系统（Cluster）的平行运算能力在近年来一直被拿出来讨论[11]。 所谓的平行运算指的是“将原本的工作分成多份，然后交给多部主机去运算，最终再将结果收集起来”的一种方式。 由于通过高速网络使用到多部主机，将能够让原本需要很长运算时间的工作，大幅的降低等待的时间！ 例如中央气象局的气象预报就很需要这样的系统来帮忙！而Linux操作系统则是这种架构下相当重要的一个环境平台呢！



Tips 由于服务器的 CPU 数量可以增加许多，而且也能够达到比较省电的功能，因此鸟哥最近更换了昆山科大资传系的模式运算服务器组， 通过 20 核心 40 超执行续的以及 12 核心 24 超执行续的两部系统，搭配 10G 网卡来处理模式的运行！用的是本书谈到的 CentOS Linux， 跑得模式是美国环保署公布，现行于世界最流行的 CMAQ 空品模式喔！

1.3.2 个人环境的使用

你知道你平时接触的电子用品中，哪些咚咚里面有Linux系统存在呢？其实相当的多呢！我们就来谈一谈吧！

- 桌面电脑：

所谓的桌面电脑，其实就是你我在办公室使用的计算机啦。一般我们称之为Desktop的系统。那么这个 Desktop 的系统平时都在做什么呢？大概都是这些工作吧：

- 上网浏览+实时通讯（Skype, FB, Google, Yahoo...）；
- 文书处理；
- 网络接口之公文处理系统；
- 办公室软件（Office Software）处理数据；
- 收发电子邮件；

想进行这些计算机工作时，你的Desktop环境需要什么咚咚？很简单，“就是需要窗口”！ 因为上网浏览、文书编排的所见即所得接口，以及电子公文系统等等， 如果没有窗口接口的辅助，那么将对使用者造成很大的困扰。而众所皆知的， Linux早期都是由工程师所发展的，对于窗口接口并没有很需要，所以造成Linux不太友好的印象。

好在，为了要强化桌面电脑的使用率，Linux与X Window System结合了！ 要注意的是，X Window System仅只是Linux上面的一套软件， 而不是核心喔！所以即使X Window挂了，对Linux也可能不会有直接的影响呢！ 更多关于 X window system 的详细信息我们留待第二十三章再来介绍。

近年来在各大社群的团结合作之下，Linux的窗口系统上面能够跑的软件实在是多的吓人！而且也能够应付的了企业的办公环境！ 例如美观的KDE与GNOME窗口接口，搭配可相容微软Office的 OpenOffice / LibreOffice （https://www.openoffice.org/zh-cn/, https://zh-cn.libreoffice.org/） 等软件， 这些自由的办公室软件包含了文书处理、电子试算表、简报软件等等，功能齐全啊！然后配合功能强大速度又快的Firefox浏览器， 以及可下载信件的雷鸟（ThunderBird）软件（类似微软的Outlook Express），还有可连上多种实时通讯的Pidgin！Linux能够做到企业所需要的各项功能啦！



Tips 鸟哥眞的垂垂老已～前一阵子 （2014） 上课时，跟学生说：“各位啊！你们考取的证照也转一份给老师来备份嘛！用 emai 寄给鸟哥喔！” 结果有几个学生竟然举手说：“老师！我知道 email 啊！不过，从来没有用过 email 寄附件耶！所以才没有传给你啊！”哇！！ 瞎密？“那你们怎么传送文件啊？用 FTP 喔？”鸟哥问，他说“没啊！就用 FB 或者是 Line 啊！或者 dropbox！眞没用过 email 耶！”...时代不同了...

- 手持系统（PDA、手机）：

自从 iphone4 在 2010 年面世之后，整个手机市场开始大搬风！智能手机市场将原本商务用的 PDA 市场整个吃掉！然后原本在 2010 年前后很热门的小笔记本也被平板电脑打趴了！在这个潮流下，Google 成立了开放手机联盟 （Open Handset Alliance） ，并且推出 Android 手机专用操作系统！ 而 Android 其实就是 Linux 核心的一支，只是专门用来针对手机/平板这类的 ARM 机器所设计的 [12]！

2015 最新的 Android 系统 5.x 使用的就是 Linux kernel 3.4.x 版本，另外，调查中也显示，从 2013 年之后，Android 系统已经是全球最多人使用的手机系统。也就是说，现在手机市场的主流操作系统是 Linux 分支出来的 Android 喔！ 那怎么能说 Linux 很少人用呢？哈哈！天天都在用耶各位！



Tips 如果你的手机是 Android 系统的话，请拿出来，然后点选“设置”--> “关于（手机）” --> “软件信息”，你就会看到 Android 版本，然后又点选“更多”， 这时你就会看到类似 3.4.10-xxx 的代号，那是什么？查一查上头提到的 Linux 版本，就知道那是啥鬼东西啰！ ^_^

- 嵌入式系统：

在第零章当中我们谈到过硬件系统， 而要让硬件系统顺利的运行就得要撰写合适的操作系统才行。那硬件系统除了我们常看到的计算机之外， 其实家电产品、PDA、手机、数码相机以及其他微型的计算机配备也是硬件系统啦！ 这些计算机配备也都是需要操作系统来控制的！而操作系统是直接嵌入于产品当中的，理论上你不应该会更动到这个操作系统， 所以就称为嵌入式系统啦！

包括路由器、防火墙、手机、IP分享器、交换器、机器人控制芯片、家电用品的微计算机控制器等等，都可以是Linux操作系统喔！ 酷学园内的Hoyo大大就曾经介绍过如何在嵌入式设备上面载入Linux！ 你桌面上用来备份的 NAS 说不定内部也是精简化过的 Linux 系统啊！

虽然嵌入式设备很多，大家也想要转而使用Linux操作系统，不过在台湾，这方面的人才还是太少了！ 要玩嵌入式系统必须要很熟悉Linux Kernel与驱动程序的结合才行！这方面的学习可就不是那么简单喔！ ^_^

1.3.3 云端运用

自从个人计算机的 CPU 内置的核心数越来越多，单一主机的能力太过强大，导致硬件资源经常闲置，这个现象让虚拟化技术得以快速发展！ 而由于硬件资源大量集中化，然后行动办公室之类的需求越来越多，因此让办公数据集中于云程序中，让企业员工仅须通过端点设备连线到云去取用运算资源， 这样就变成无时无地都可以办公啦 （其实很惨...永远不得休息啊！眞可怜～）！

这就是三国演义里面谈到的“天下大势，分久必合、合久必分”的名言啊！ 从（1）早期的贵森森的大型主机分配数个终端机的集中运算机制，到 （2）2010 年前个人计算机运算能力增强后，大部分的运算都是在台式机或笔记本上自行达成， 再也不需要跑去大型主机取得运算资源了！到现在（3）由于行动设备的发达，产生的庞大数据需要集中处理，因而产生云端系统的需求！让信息/资源集中管理！ 这不是分分合合的过程吗？^_^

- 云程序

许多公司都有将资源集中管理的打算，之前参与一场座谈会，有幸遇到阿里巴巴的架构师，鸟哥偷偷问他说，他们机房里面有多少计算机主机啊？ 他说不多！差不多才 2 万部主机而已...鸟哥正在搞的可提供 200 个左右的虚拟机的系统，使用大约 7 部主机就觉得麻烦了，他们家至少有 2 万部耶！ 这么多的设备底层使用的就是 Linux 操作系统来统一管理。

另外，除了公司自己内部的私有云之外，许多大型网际网络供应商 （ISP） 也提供了所谓的公有云来让企业用户或个人用户来使用 ISP 的虚拟化产品。 因此，如果公司内部缺乏专业管理维护人才，很有可能就将自家所需要的关键应用如 Web、Mail、系统开发环境等操作系统交由 ISP 代管， 自家公司仅须远端登陆该系统进行网站内容维护或程序开发而已。那这些虚拟化后的系统，也经常是 Linux 啊！因为跟上头企业环境利用提到的功能是相同的！

所以说云程序的底层就是 Linux ，而云程序搭建出来的虚拟机，内容也是 Linux 操作系统哩！用的越来越多啊！



Tips 所谓的“虚拟化”指的是：在一部实体主机上面仿眞出多个逻辑上完全独立的硬件，这个假的虚拟出来的硬件主机，可以用来安装一部逻辑上完全独立的操作系统！ 因此，通过虚拟化技术，你可以将一部实体主机安装多个同时运行的操作系统 （非多重开机），以达到将硬件资源完整利用的效果。 很多 ISP 就是通过贩售这个虚拟机的使用权来赚钱的喔！

- 端设备

既然运算资源都集中在云里面了，那我需要连线到云程序的设备应该可以越来越轻量吧？当然没错！所以智能手机才会这么热门啊！ 很多时候你只要有智能手机或者是平板，连线到公司的云里面去，就可以开始办公了哩！

此外，还有更便宜的端点设备喔！那就是近年来很热门又流行的树莓派（Raspberry Pi）与香蕉派（Banana Pi），这两个小东西售价都不到 50 美元，有的甚至台币 1000 块有找！这个 Raspberry Pi 其实就是一部小型的计算机，只要加上 USB 键盘、鼠标与 HDMI 的屏幕，立刻就是可以让小朋友学习程序语言的环境！如果加上通过网络去取得具有更强大运算资源的云端虚拟机，不就可以做任何事了吗？所以，端点设备理论上会越来越轻量化的！



Tips 鸟哥近几年来做的主要研究，就是通过一组没很贵的 server 系统达成打开多个虚拟机的环境，然后让学生可以在教室利用类似 banana pi 的设备来连线到服务器，这时学生就可以通过网络来取得一套完整的操作系统，可以拿来上课、回家实作练习、上机考试等等！相当有趣！鸟哥称为虚拟计算机教室！而 server 与 banana pi 的内部操作系统当然就是 Linux 啊！

1.4 Linux 该如何学习

为什么大家老是建议学习Linux最好能够先舍弃X Window的环境呢？ 这是因为X window了不起也只是Linux内的“一套软件”而不是“Linux核心”。 此外，目前发展出来的X-Window对于系统的管理上还是有无法掌握的地方，举个例子来说，如果 Linux本身捉不到网卡的时候，请问如何以X Window来捉这个硬件并且驱动他呢？

还有，如果需要以Tarball（源代码）的方式来安装软件并加以设置的时候，请以X Window 来架设他！这可能吗？当然可能，但是这是在考验“X Window开发商”的技术能力， 对于了解Linux架构与核心并没有多大的帮助的！所以说，如果只是想要“会使用Linux”的角度来看， 那么确实使用X Window也就足够了，反正搞不定的话，花钱请专家来搞定即可；但是如果想要更深入 Linux的话，那么命令行界面才是不二的学习方式！

以服务器或者是嵌入式系统的应用来说，X Window是非必备的软件，因为服务器是要提供用户端来连线的， 并不是要让使用者直接在这部服务器前面按键盘或鼠标来操作的！所以图形接口当然就不是这么重要了！ 更多的时候甚至大家会希望你不要启动X window在服务器主机上，这是因为X Window通常会吃掉很多系统资源的缘故！

再举个例子来说，假如你是个软件服务的工程师，你的客户人在台北，而你人在远方的台南。某一天客户来电说他的Linux服务器出了问题， 要你马上解决他，请问：要您亲自上台北去修理？还是他搬机器下来让你修理？或者是直接请他开个帐号给你进去设置即可？ 想当然尔，就会选择开帐号给你进入设置即可啰！因为这是最简单而且迅速的方法！这个方法通常使用命令行会较为单纯， 使用图形接口则非常麻烦啦！所以啦！这时候就得要学学命令行来操作Linux比较好啦！

另外，在服务器的应用上，文件的安全性、人员帐号的管理、软件的安装/修改/设置、 登录文件的分析以及自动化工作调度与程序的撰写等等，都是需要学习的， 而且这些东西都还未涉及服务器软件呢！对吧！这些东西真的很重要，所以，建议你得要依据下面的介绍来学习才好。



Tips 这里是站在要让 Linux 成为自己的好用的工具 （服务器或开发软件的程序学习平台） 为出发点去介绍如何学习的喔！所以，不要以旧有的 Windows 角度来思考！ 也不要说“你都只有碰过触摸式设备”的角度来思考！加油啰！

1.4.1 从头学习Linux基础

其实，不论学什么系统，“从头学起”是很重要的！还记得你刚刚接触微软的Windows都在干什么？ 还不就是由文件资源管理器学起，然后慢慢的玩到控制台、玩到桌面管理， 然后还去学办公室软件，我想，你总该不会直接就跳过这一段学习的历程吧？那么 Linux的学习其实也差不多，就是要从头慢慢的学起啦！不能够还不会走路之前就想要学飞了吧！^_^！

常常有些朋友会写信来问鸟哥一些问题，不过，信件中大多数的问题都是很基础的！ 例如：“为什么我的使用者个人网页显示我没有权限进入？”、“为什么我下达一个指令的时候，系统告诉我找不到该指令？”、“我要如何限制使用者的权限”等等的问题，这些问题其实都不是很难的，只要了解了 Linux的基础之后，应该就可以很轻易的解决掉这方面的问题呢！ 所以请耐心的，慢慢的，将后面的所有章节内容都看完。自然你就知道如何解决了！

此外，网络基础与安全也很重要，例如TCP/IP的基础知识，网络路由的相关概念等等。 很多的朋友一开始问的问题就是“为什么我的邮件服务器主机无法收到信件？” 这种问题相当的困扰，因为发生的原因太多了，而朋友们常常一接触Linux就是希望 “建站！”根本没有想到要先了解一下Linux的基础！这是相当伤脑筋的！ 尤其近来计算机怪客（Cracker）相当多，（真奇怪，闲闲没事干的朋友还真是不少....） ，一个不小心您的主机就被当成怪客跳板了！ 甚至发生被警告的事件也层出不穷！这些都是没能好好的注意一下网络基础的原因呀！

所以，鸟哥希望大家能够更了解Linux，好让他可以为你做更多的事情喔！ 而且这些基础知识是学习更深入的技巧的必备条件呀！因此建议：

1. 计算机概论与硬件相关知识： 因为既然想要走Linux这门路，信息相关的基础技能也不能没有啊！ 所以先理解一下基础的硬件知识，不用一定要全懂啦！又不是真的要你去组计算机～^_^， 但是至少要“听过、有概念”即可；

2. 先从Linux的安装与指令学起： 没有Linux怎么学习Linux呢？所以好好的安装起一套你需要的Linux吧！虽然说Linux distributions很多， 不过基本上架构都是大同小异的，差别在于接口的友好度与软件的选择不同罢了！ 选择一套你喜欢的就好了，倒是没有哪一套特别好说～

3. Linux操作系统的基础技能： 这些包含了“使用者、群组的概念”、“权限的观念”，“程序的定义”等等，尤其是权限的概念， 由于不同的权限设置会妨碍你的使用者的便利性，但是太过于便利又会导致入侵的可能！ 所以这里需要了解一下你的系统呦！

4. 务必学会vi文书编辑器： Linux的文书编辑器多到会让你数到生气！不过，vi却是强烈建议要先学习的！ 这是因为vi会被很多软件所调用，加上所有的Unix like系统上面都有vi，所以你一定要学会才好！

5. Shell与Shell Script的学习： 其实鸟哥上面一直谈到的“命令行”说穿了就是一个名为shell的软件啦！既然要玩命令行，当然就是要会使用shell的意思。 但是shell上面的数据太多了，包括“正则表达式”、“管线命令”与“数据流重导向”等等，真的需要了解比较好呦！ 此外，为了帮助你未来的管理服务器的便利性，shell scripts也是挺重要的！要学要学！

6. 一定要会软件管理员： 因为玩Linux常常会面临得要自己安装驱动程序或者是安装额外软件的时候，尤其是嵌入式设备或者是学术研究单位等。 这个时候Tarball/RPM/DPKG/YUM/APT等软件管理员的安装方式的了解，对你来说就重要到不行了！
7. 网络基础的创建： 如果上面你都通过了，那么网络的基础就是下一阶段要接触的咚咚，这部份包含了“IP概念”“路由概念”等等；
8. 如果连网络基础都通过了，那么网站的架设对你来说，简直就是“太简单啦！”

在一些基础知识上，可能的话，当然得去书店找书来读啊！ 如果您想要由网络上面阅读的话，那么这里推荐一下由Netman大哥评论员的Study-Area里面的基础文章，相当的实用！

- 计算机基础 （http://www.study-area.org/compu/compu.htm）
- 网络基础 （http://www.study-area.org/network/network.htm）

1.4.2 选择一本易读的工具书

正所谓这：“好的书本带你上天堂、坏的书本让你穷瞎忙...”一本好的工具书是需要的，不论是未来作为查询之用，还是在正确的学习方法上。 可惜的是，目前坊间的书大多强调速成的Linux教育，或者是强调Linux的网络功能， 却欠缺了大部分的Linux基础管理～鸟哥在这里还是要再次的强调，Linux的学习历程并不容易， 他需要比较长的时间来适应、学习与熟悉，但是只要能够学会这些简单的技巧， 这些技巧却可以帮助您在各个不同的OS之间遨游！

您既然看到这里了，应该是已经取得了鸟哥的 Linux 私房菜 -- 基础学习篇了吧！ ^_^ 。 希望这本书可以帮助您缩短基础学习的历程，也希望能够带给您一个有效的学习观念！ 而在这本书看完之后，或许还可以参考一下Netman推荐的相关网络书籍：

- 请推荐有关网络的书：
  http://linux.vbird.org/linux_basic/0120howtolinux/0120howtolinux_1.php

不过，要强调的是，每个人的阅读习惯都不太一样，所以，除了大家推荐的书籍之外， 您必须要亲眼看过该本书籍，确定您可以吸收的了书上的内容，再下去购买喔！



Tips 其实鸟哥买科技类书籍比较喜欢买基础书耶，因为基础学好了，其他的部份大概找个keyword ，再 google 一下，一大堆数据就可以让你去分析判断了！ 你会说，既然如此，那基础书籍内的项目不是 google 也是一大堆？不要忘记了，“最开始你是要用什么关键字去google 啊？”！ 所以，阅读基础书籍的重点，就是让自己能够掌握住那些“ keyword ”啰！加油！

1.4.3 实作再实作

要增加自己的体力，就是只有运动；要增加自己的知识，就只有读书；当然，要增加自己对于Linux的认识， 大概就只有实作经验了！所以，赶快找一部计算机，赶快安装一个Linux distribution，然后快点进入Linux的世界里面晃一晃！ 相信对于你自己的Linux能力必然大有斩获！除了自己的实作经验之外，也可以参考网络上一些善心人士整理的实作经验分享喔！ 例如最有名的Study-Area（http://www.study-area.org）等网站。

此外，人脑不像计算机的硬盘一样，除非硬盘坏掉了或者是数据被你抹掉了， 否则储存的数据将永远而且立刻的记忆在硬盘中！在人类记忆的曲线中， 你必须要“不断的重复练习”才会将一件事情记得比较熟！同样的，学习Linux也一样， 如果你无法经常摸索的话，那么，抱歉的是，学了后面的， 前面的忘光光！学了等于没学，这也是为什么鸟哥当初要写“鸟哥的私房菜”这个网站的主要原因， 因为，鸟哥的忘性似乎比一般人还要好～～呵呵！所以，除了要实作之外，还得要常摸！才会熟悉Linux而且不会怕他呢！



Tips 鸟哥上课时，常常有学生问到：“老师，到底要听过你的课几次之后，才能学的会？”鸟哥的标准答案是：“你永远学不会！” 因为你是用“听”的，没有动手做，那么永远不会知道“经验”两个字怎么写！很多时候计算机/网络都会有一些莫名其妙的突发状况， 没有实际碰触过，怎么可能会理解呢？所以“永远是不可能听会的！”为啥要实验？因为实验过后你才会有经验来记下来？ 否则实验结果课本都有啊！不是背一背就好了，干嘛实验呢？浪费钱吗？ ^_^

1.4.4 发生问题怎么处理啊？建议流程是这样...

我们是“人”不是“神”，所以在学习的过程中发生问题是很常见的啦！重点是，我们该如何处理在自身所发生的Linux问题呢？ 在这里鸟哥的建议是这样的流程：

1. 在自己的主机/网络数据库上查询How-To或FAQ

其实，在Linux主机及网络上面已经有相当多的FAQ整理出来了！所以，当你发生任何问题的时候，除了自己检查， 或者到上述的实作网站上面查询一下是否有设置错误的问题之外，最重要的当然就是到各大FAQ的网站上查询啰！ 以下列出一些有用的FAQ与How-To网站给您参考一下：

- Linux自己的文件数据： /usr/share/doc （在你的Linux系统中）
- CLDP 中文文件计划 http://www.linux.org.tw/CLDP/
- The Linux Documentation Project：http://www.tldp.org/

上面比较有趣的是那个TLDP（The Linux Documentation Project），他几乎列出了所有Linux上面可以看到的文献数据，各种How-To的作法等等，虽然是英文的，不过，很有参考价值！

除了这些基本的FAQ之外，其实，还有更重要的问题查询方法，那就是利用酷狗（Google）帮您去搜寻答案呢！在鸟哥学习Linux的过程中，如果有什么奇怪的问题发生时，第一个想到的，就是去http://www.google.com.tw搜寻是否有相关的议题。举例来说，我想要找出Linux下面的NAT，只要在上述的网站内，输入Linux跟NAT，立刻就有一堆文献跑出来了！真的相当的优秀好用喔！您也可以通过酷狗来找鸟哥网站上的数据呢！

- Google：http://www.google.com.tw
- 鸟哥网站：http://linux.vbird.org/Searching.php

- 注意讯息输出，自行解决疑难杂症：

一般而言，Linux在下达指令的过程当中，或者是log file里头就可以自己查得错误信息了，举个例子来说，当你下达：

```
[root@centos ~]# ls -l /vbird
```

由于系统并没有 /vbird 这个目录，所以会在屏幕前面显示：

```
ls: /vbird: No such file or directory
```

这个错误讯息够明确了吧！系统很完整的告诉您“查无该数据”！呵呵！所以啰，请注意，发生错误的时候，请先自行以屏幕前面的信息来进行 debug（除错）的动作，然后，如果是网络服务的问题时，请到/var/log/这个目录里头去查阅一下 log file（登录文件），这样可以几乎解决大部分的问题了！

1. 搜寻过后，注意网络礼节，讨论区大胆的发言吧：

一般来说，如果发生错误现象，一定会有一些讯息对吧！那么当您要请教别人之前，就得要将这些讯息整理整理，否则网络上人家也无法告诉您解决的方法啊！这一点很重要的喔！

万一真的经过了自己的查询，却找不到相关的信息，那么就发问吧！不过，在发问之前建议您最好先看一下“ 提问的智慧 http://phorum.vbird.org/viewtopic.php?t=96” 这一篇讨论！然后，你可以到下面几个讨论区发问看看：

- 酷学园讨论区 http://phorum.study-area.org
- 鸟哥的私房菜馆讨论区 http://phorum.vbird.org

不过，基本上去每一个讨论区回答问题的熟手，其实都差不多是那几个，所以，您的问题“不要重复发表在各个主要的讨论区！”举例来说，鸟园与酷学园讨论区上的朋友重复性很高，如果您两边都发问，可能会得到反效果，因为大家都觉得，另外一边已经回答您的问题了呢～～

1. Netman大大给的建议：

此外，Netman 兄提供的一些学习的基本方针，提供给大家参考：

- 在Windows里面，程序有问题时，如果可能的话先将所有其它程序保存并结束，然后尝试按救命三键 （Ctrl+Alt+Delete），将有问题的程序（不要选错了程序哦）“结束工作”，看看能不能恢复系统。不要动不动就直接关机或reset。
- 有系统地设计文件目录，不要随便到处保存盘案以至以后不知道放哪里了， 或找到文件也不知道为何物。
- 养成一个做记录的习惯。尤其是发现问题的时候， 把错误信息和引发状况以及解决方法记录清楚，同时最后归类及定期整理。别以为您还年轻，等你再弄多几年计算机了， 您将会非常庆幸您有此一习惯。
- 如果看在网络上看到任何好文章，可以为自己留一份copy，同时定好题目，归类存盘。（鸟哥需要注意知识产权！）
- 作为一个使用者，人要迁就机器；做为一个开发者，要机器迁就人。
- 学写 script 的确没设置 server 那么好玩，不过以我自己的感觉是：关键是会得“偷”， 偷了会得改，改了会得变，变则通矣。
- 在Windows里面，设置不好设备，您可以骂它；在Linux里面，如果设置好设备了，您得要感激它！

1.4.5 鸟哥的建议（重点在solution的学习）

除了上面的学习建议之外，还有其他的建议吗？确实是有的！其实， 无论作什么事情，对人类而言，两个重要的因素是造成我们学习的原动力：

- 成就感
- 兴趣

很多人问过我，鸟哥是怎么学习Linux的？由上面鸟哥的悲惨Linux学习之路你会发现， 原来我本人对于计算机就蛮有兴趣的，加上工作的需要，而鸟哥又从中得到了相当多的成就感，所以啰，就一发不可收十的爱上Linux啰！因此，鸟哥个人认为，学习Linux如果玩不出兴趣，他对你也不是什么重要的生财工具，那么就不要再玩下去了！ 因为很累人ㄋㄟ～而如果你眞的想要玩这么一套优良的操作系统， 除了前面提到的一些建议之外，说眞的，得要培养出兴趣与成就感才行！ 那么如何培养出兴趣与成就感呢？可能有几个方向可以提供给你参考：

- 创建兴趣： Linux上面可以玩的东西眞的太多了，你可以选择一个有趣的课题来深入的玩一玩！不论是Shell还是图形接口等等， 只要能够玩出兴趣，那么再怎么苦你都会不觉得喔！

- 成就感： 成就感是怎么来的？说实在话，就是“被认同”来的！怎么被认同呢？写心得分享啊！当你写了心得分享，并且公告在 BBS 上面，自然有朋友会到你的网页去瞧一瞧，当大家觉得你的网页内容很棒的时候， 哈哈！你肯定会加油继续的分享下去而无法自拔的！那就是我啦...... ^_^！

就鸟哥的经验来说，你“学会一样东西”与 “要教人家会一样东西”思考的纹路是不太一样的！ 学会一样东西可能学一学会了就算了！但是要“教会”别人，那可就不是闹着玩的！得要思考相当多的理论性与实务性方面的咚咚，这个时候，你所能学到的东西就更深入了！ 鸟哥常常说，我这个网站对我在Linux的了解上面真的的帮助很大！

- 协助回答问题： 另一个创造成就感与满足感的方法就是“助人为快乐之本！”当你在 BBS 上面告诉一些新手，回答他们的问题，你可以获得的可能只是一句“谢谢！感恩呐！” 但是那句话真的会让人很有快乐的气氛！很多的老手都是因为有这样的满足感， 才会不断的协助新来的朋友的呢！此外，回答别人问题的时候，就如同上面的说明一般， 你会更深入的去了解每个项目，哈哈！又多学会了好多东西呢！
- 参与讨论： 参与大家的技术讨论一直是一件提升自己能力的快速道路！因为有这些技术讨论， 你提出了意见，不论讨论的结果你的意见是对是错，对你而言，都是一次次的知识成长！这很重要喔！ 目前台湾地区办活动的能力是数一数二的Linux社群“酷学园（Study Area, SA）”，每个月不定期的在北/中/南举办自由软件相关活动， 有兴趣的朋友可以看看： http://phorum.study-area.org/index.php/board,22.0.html

除了这些基本的初学者建议外，其实，对于未来的学习，这里建议大家要“眼光看远！”一般来说，公司行号会发生问题时， 他们绝不会只要求各位“单独解决一部主机的问题”而已，他们需要的是整体环境的总体解决“Total Solution”。 而我们目前学习的Linux其实仅是在一部主机上面进行各项设置而已， 还没有到达解决整体公司所有问题的状态。当然啦，得要先学会Linux相关技巧后， 才有办法将这些技巧用之于其他的solution上面！

所以，大家在学习Linux的时候，千万不要有“门户之见”，认为MS的东西就比较不好～ 否则，未来在职场上，竞争力会比人家弱的！有办法的话，多接触，不排斥任何学习的机会！都会带给自己很多的成长！ 而且要谨记：“不同的环境下，解决问题的方法有很多种，只要行的通，就是好方法！”



Tips 另外，不要再说没兴趣了！没有花时间去了解一下，不要跟人家说你没兴趣！而且，兴趣也是靠培养来的！除了某些特殊人物之外， 没有花时间趣培养兴趣，怎么可能会有兴趣！？

1.5 重点回顾

- 操作系统（Operation System）主要在管理与驱动硬件，因此必须要能够管理内存、管理设备、 负责行程管理以及系统调用等等。因此，只要能够让硬件准备妥当（Ready）的情况， 就是一个阳春的操作系统了。
- Unix的前身是由贝尔实验室（Bell lab.）的Ken Thompson利用组合语言写成的， 后来在1971-1973年间由Dennis Ritchie以C程序语言进行改写，才称为Unix。
- 1977年由Bill Joy释出BSD （Berkeley Software Distribution），这些称为Unix-like的操作系统。
- 1984年由Andrew Tanenbaum开始制作Minix操作系统，该系统可以提供源代码以及软件；
- 1984年由Richard Stallman提倡GNU计划，倡导自由软件（Free software）， 强调其软件可以"自由的取得、复制、修改与再发行"，并规范出GPL授权模式， 任何GPL（General Public License）软件均不可单纯仅贩卖其软件，也不可修改软件授权。
- 1991年由芬兰人Linus Torvalds开发出Linux操作系统。简而言之，Linux成功的地方主要在于： Minix（Unix）, GNU, Internet, POSIX 及虚拟团队的产生。
- 符合 Open source 理念的授权相当多，比较知名的如 Apache / BSD / GPL / MIT 等。
- Linux本身就是个最阳春的操作系统，其开发网站设立在http://www.kernel.org，我们亦称Linux操作系统最底层的数据为"核心（Kernel）"。
- 从 Linux kernel 3.0 开始，已经舍弃奇数、偶数的核心版本规划，新的规划使用主线版本（MainLine） 为依据， 并提供长期支持版本 （longterm） 来加强某些功能的持续维护。
- Linux distributions的组成含有："Linux Kernel + Free Software + Documentations（Tools） + 可完整安装的程序"所制成的一套完整的系统。
- 常见的 Linux distributions 分类有"商业、社群"分类法，或"RPM、DPKG"分类法
- 学习 Linux 最好从头由基础开始学习，找到一本适合自己的书籍，加强实作才能学会

1.6 本章习题

（要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看） 实作题部分：

- 请上网找出目前 Linux 核心的最新稳定版与发展中版本的版本号码，请注明查询的日期与版本的对应。
- 请上网找出 Linux 的吉祥物企鹅的名字，以及最原始的图像文件画面。（提示：请前往 http://www.linux.org 查阅）
- 请上网找出 Andriod 与 Linux 核心版本间的关系。（提示：请前往 https://zh.wikipedia.org/wiki/Android 查阅）

简答题部分：

- 你在你的主机上面安装了一张网卡，但是开机之后，系统却无法使用，你确定网卡是好的，那么可能的问题出在哪里？该如何解决？因为所有的硬件都没有问题，所以，可能出问题的地方在于系统的核心（kernel） 不支持这张网卡。解决的方法，（1）到网卡的开发商网站，（2）下载支持你主机操作系统的驱动程序，（3）安装网卡驱动程序后，就可以使用了。
- 一个操作系统至少要能够完整的控制整个硬件，请问，操作系统应该要控制硬件的哪些单元？根据硬件的运行，以及数据在主机上面的运算情况与写入/读取情况，我们知道至少要能够控制：（1）input/output control,（2）device control,（3）process management,（4）file management. 等等！
- 我在Windows上面玩的游戏，可不可以拿到Linux去玩？当然不行！因为游戏也是一个应用程序（application），他必须要使用到核心所提供的工具来开发他的游戏，所以这个游戏是不可在不同的平台间运行的。除非这个游戏已经进行了移植。
- Linux本身仅是一个核心与相关的核心工具而已，不过，他已经可以驱动所有的硬件，所以，可以算是一个很阳春的操作系统了。经过其他应用程序的开发之后，被整合成为Linux distribitions。请问众多的distributions之间，有何异同？相同：（1）同样使用 http://www.kernel.org 所释出的核心；（2）支持同样的标准，如 FHS、LSB 等；（3）使用几乎相同的自由软件（例如 GNU 里面的 gcc/glibc/vi/apache/bind/sendmail...）；（4）几乎相同的操作接口（例如均使用 bash/KDE/GNOME 等等）。 不同：使用的 kernel 与各软件的版本可能会不同；各开发商加入的应用工具不同，使用的套件管理模式不同（dpkg 与 RPM）
- Unix 是谁写出来的？ GNU 计划是谁发起的？Unix 是 Ken Thompson 写的，1973 年再由 Dennis Ritchie 以 C 语言改写成功。 至于 GNU 与 FSF 则是 Richard Stallman 发起的。
- GNU 的全名为何？他主要由那个基金会支持？GNU 是 GNU is Not Unix 的简写，是个无穷循环！ 另外，这个计划是由自由软件基金会（Free Software Foundation, FSF） 所支

持的！ 两者都是由 Stallman 先生所发起的！

- 何谓多用户 （ Multi-user ） 多任务 （ Multitask ）？Multiuser 指的是 Linux 允许多人同时连上主机之外，每个使用者皆有其各人的使用环境，并且可以同时使用系统的资源！Multitask 指的是多任务环境，在 Linux 系统下， CPU 与其他例如网络资源可以同时进行多项工作， Linux 最大的特色之一即在于其多任务时，资源分配较为平均！
- 简单说明 GNU General Public License （ GPL ） 与 Open Source 的精神：1. GPL 的授权之软件，乃为自由软件（Free software），任何人皆可拥有他； 2. 开发 GPL 的团体（或商业企业）可以经由该软件的服务来取得服务的费用； 3. 经过 GPL 授权的软件，其属于 Open source 的情况，所以应该公布其源代码； 4. 任何人皆可修改经由 GPL 授权过的软件，使符合自己的需求； 5. 经过修改过后 Open source 应该回馈给 Linux 社群。
- 什么是 POSIX ?为何说 Linux 使用 POSIX 对于发展有很好的影响？POSIX 是一种标准规范，主要针对在 Unix 操作系统上面跑的程序来进行规范。 若你的操作系统符合 POSIX ，则符合 POSIX 的程序就可以在你的操作系统上面运行。 Linux 由于支持 POSIX ，因此很多 Unix 上的程序可以直接在 Linux 上运行， 因此程序的移植相当简易！也让大家容易转换平台，提升 Linux 的使用率。
- 简单说明 Linux 成功的因素？1. 借由 Minix 操作系统开发的 Unix like ，没有版权的纠纷； 2. 借助于 GNU 计划所提供的各项工具软件， gcc/bash 等； 3. 借由 Internet 广为流传； 4. 借由支持 POSIX 标准，让核心能够适合所有软件的开发； 5. 托瓦兹强调务实，虚拟团队的自然形成！

1.7 参考资料与延伸阅读

- [1]Multics计划网站：http://www.multicians.org/。
- [2]Ken Thompson 的 wiki 简介：http://en.wikipedia.org/wiki/Ken_Thompson
- [3]Dennis Ritchie 的 wiki 简介： http://en.wikipedia.org/wiki/Dennis_Ritchie
- [4]Bill joy 的 wiki 简介： http://en.wikipedia.org/wiki/Bill_Joy
- [5]Andrew Tanenbaum 的 wiki 简介：http://en.wikipedia.org/wiki/Andrew_S._Tanenbaum
- [6]Richard Stallman的个人网站： http://www.stallman.org/
- [7]GNU 计划的官网： http://www.gnu.org/
- [8]开放源代码促进会针对 open source 的解释： http://opensource.org/definition 以及 Open source 与 free software 的差异：http://opensource.org/faq#free-software
- [9]开放源代码促进会针对 Open source 授权的汇整介绍：http://opensource.org/licenses
- [10]Linus Torvalds 在 Wiki 的介绍： http://en.wikipedia.org/wiki/Linus_Torvalds
- [11]Cluster Computer 在 Wiki 的介绍： http://en.wikipedia.org/wiki/Computer_cluster
- [12]Android 在 Wiki 的介绍： http://zh.wikipedia.org/wiki/Android
- 洪朝贵老师的GNU/FSF介绍： http://people.ofset.org/~ckhung/a/c_83.php
- 葛林穆迪着，杜默译，“Linux传奇”，时报文化出版企业。 书本介绍：http://findbook.tw/book/9789571333632/basic
- XFree86的网站：http://www.xfree86.org/
- POSIX的相关说明： 维基百科：http://en.wikipedia.org/wiki/POSIX IEEE POSIX标准：http://standards.ieee.org/regauth/posix/

2002/06/25：第一次完成 2003/01/26：重新修订，加入一些历史事件、重新编排与加入 FAQ 2003/02/28：加入百资以及 distrowatch 两个网站的推荐！ 2005/05/31：旧有的数据放于 此处 2005/06/02：做了大幅度的改版，很多数据参考了网络农夫及 Linux 传奇等书籍，建议大家要多看看网络农夫的大作喔！ 2005/06/08：将原本的 binary / compiler / Emacs 的地方再说明一下！比较容易了解那是什么！顺便加入习题 2005/07/21：网络农夫的网站结束了～真伤心～只好提供网络农夫之前发表的文章链接了！ 2005/08/03：感谢网友 babab 的来信告知，修订了国家高速网络中心网址：http://www.nchc.org.tw 2005/10/24：经由网友的回报，洪朝贵老师已经调职到树德大学，因此整个链接内容已作修订。 2006/05/31：加入了重点回顾的项目啦！ 2006/06/06：感谢网友 "Warren Hsieh" 兄的提醒，由于MAC在 2006 年后使用 Intel 的 x86 硬件架构，故 Windows 是可能可以在上面安装的！ 2008/07/23：因为加入了计算机概论的章节，所以本文做了挺大幅度的修改！原本针对FC4的版本请点选这里。 2007/07/26：将整份文章重新校阅过，修订一些文辞，也将格式调整为适合的XHTML了！ 2007/07/29：将主、次核心版本加强说明！ 2009/08/05：移除最后一小节的标准，将FHS与

LSB向前挪到distribution解释中。拿掉服务器、工作站、终端机的说明。 2012/02/20：更新了 Linux 在台湾的相关链接信息 2015/04/17：旧的基于 CentOS 5 的数据放置在这里喔！ 2015/04/23：同时整合了Linux 如何学习这一章的内容！减少一些不必要的碎碎念

第二章、主机规划与磁盘分区

最近更新日期：20//

事实上，要安装好一部Linux主机并不是那么简单的事情，你必须要针对distributions的特性、服务器软件的能力、 未来的升级需求、硬件扩充性需求等等来考虑，还得要知道磁盘分区、文件系统、Linux操作较频繁的目录等等， 都得要有一定程度的了解才行，所以，安装Linux并不是那么简单的工作喔！ 不过，要学习Linux总得要有Linux系统存在吧？所以鸟哥在这里还是得要提前说明如何安装一部Linux练习机。 在这一章里面，鸟哥会介绍一下，在开始安装Linux之前，您应该要先思考哪些工作？ 好让您后续的主机维护轻松愉快啊！此外，要了解这个章节的重要性，您至少需要了解到Linux文件系统的基本概念， 这部份初学者是不可能具备的！所以初学者在这个章节里面可能会觉得很多部份都是莫名其妙！没关系！ 在您完成了后面的相关章节之后，记得要再回来这里看看如何规划主机即可！ ^_^

2.1 Linux与硬件的搭配

虽然个人计算机各元件的主要接口是大同小异的，包括前面第零章计算机概论讲到的种种接口等， 但是由于新的技术来得太快，Linux核心针对新硬件所纳入的驱动程序模块比不上硬件更新的速度， 加上硬件厂商针对Linux所推出的驱动程序较慢，因此你在选购新的个人计算机（或服务器）时， 应该要选择已经过安装Linux测试的硬件比较好。

此外，在安装Linux之前，你最好了解一下你的Linux预计是想达成什么任务，这样在选购硬件时才会知道那个元件是最重要的。 举例来说，桌面电脑（Desktop）的使用者，应该会用到X Window系统， 此时，显卡的优劣与内存的大小可就占有很重大的影响。如果是想要做成文件服务器， 那么硬盘或者是其他的储存设备，应该就是您最想要增购的元件啰！所以说，功课还是需要作的啊！

鸟哥在这里要不厌其烦的再次的强调，Linux对于计算机各元件/设备的分辨， 与大家惯用的Windows系统完全不一样！因为，各个元件或设备在Linux下面都是“一个文件！” 这个观念我们在第一章Linux是什么里面已经提过， 这里我们再次的强调。因此，你在认识各项设备之后，学习Linux的设备文件名之前， 务必要先将Windows对于设备名称的概念先拿掉～否则会很难理解喔！

2.1.1 认识计算机的硬件配备

“什么？学Linux还得要玩硬件？”呵呵！没错！ 这也是为什么鸟哥要将计算机概论搬上台面之故！ 我们这里主要是介绍较为普遍的个人计算机架构来设置Linux服务器，因为比较便宜啦！至于各相关的硬件元件说明已经在第零章计概内讲过了，这里不再重复说明。 仅将重要的主板与元件的相关性图示如下：

图2.1.1、个人计算机各元件的相关性 （上述图示主要取自tom's硬件指南，各元件图片分属个别公司所有）

那么我们应该如何挑选计算机硬件呢？随便买买就好，还是有特殊的考虑？下面有些思考角度可以提供给大家参考看看：

- 游戏机/工作机的考虑

事实上，计算机主机的硬件配备与这部主机未来的功能是很有相关性的！举例来说，家里有小孩， 或者自己仍然算是小孩的朋友大概都知道：“要用来打Game的“游戏机计算机” 所需要的配备一定比办公室用的“工作机计算机”配备更高档”，为什么呢？ 因为现在一般的三维（3D）计算机游戏所需要的3D光影运算太多了，所以显卡与CPU资源都会被耗用的非常多！当然就需要比较高级的配备啰，尤其是在显卡、CPU（例如Intel的 I5, I7 系列的） 及主板芯片组方面的功能。

至于办公室的工作环境中，最常使用到的软件大多是办公软件（Office），最常使用的网络功能是浏览器， 这些软件所需要的运算并不高，理论上目前的入门级计算机都能够跑得非常顺畅了！ 甚至很多企业都喜欢购买将显卡、主板芯片组整合在一起的整合型芯片的计算机，因为便宜又好用！

- “性能/价格”比与“性能/消耗的瓦数”比的考虑

并不是“贵就比较好”喔！在目前（2015）电费居高不下的情况，如何兼顾省钱与计算机硬件的性能问题，很重要！ 如果你喜欢购买最新最快的计算机零件，这些刚出炉的元件都非常的贵，而且操作系统还不见得能够完整的支持。 所以，鸟哥都比较喜欢购买主流级的产品而非最高档的。因为我们最好能够考虑到性能/价格比。 如果高一级的产品让你的花费多一倍，但是新增加的性能却只有10%而已，那这个性能/价格的比值太低，不建议啦！

此外，由于电价越来越高，如何“省电”就很重要啦！因此目前硬件评论界有所谓的“每瓦性能”的单位， 每瓦电力所发挥的性能越高，当然代表越省电啊！这也是购买硬件时的考虑之一啦！要知道，如果是做为服务器用， 一年365天中时时刻刻都开机，则你的计算机多花费50瓦的电力时，每年就得要多花450度电左右（50W*365*天24小时/天/1000W=438度电），如果以企业来讲， 每百部计算机每年多花450度电的话，每年得多花十万块以上的电费呢 （以一度电3块钱来计算） ！所以这也需要考虑啊！

- 支持度的考虑

并非所有的产品都会支持特定的操作系统，这牵涉到硬件开发商是否有意愿提供适当的驱动程序之故。 因此，当我们想要购买或者是升级某些计算机元件时，应该要特别注意该硬件是否有针对您的操作系统提供适当的驱动程序， 否则，买了无法使用，那才是叫人呕死啊！因此，针对Linux来说，下面的硬件分析就重要啦！



Tips因为鸟哥会自己编译驱动程序，所以上次买家用桌面电脑时，就委托鸟嫂全权处理 （因为钱钱是鸟嫂负责的嘛！嘿嘿！省的麻烦！）！ 反正最多就是自己去找 driver 来编译，那也没什么～您说是吧？ 没想到来的主板上面内置的那颗网卡驱动程序，网卡开发商的官网上面并没有提供 source code！鸟哥赶紧回去查一下该主板的说明， 结果...说明书上面明明白白的说，这块主板仅提供支持 windows 的 drivers 而已...还建议不要拿来装 Linux 之用... 当下还是默默的去找了一块 PCI-e 网卡来插了...连 source code 都没有，是要编译啥啦！巧妇难为无米之炊啊～～ @_@～～ 这个故事告诉我们，作人不要太铁齿，硬件该查阅的工作还是要做啦！

2.1.2 选择与Linux搭配的主机配备

由于硬件的加速发展与操作系统核心功能的增强，导致较早期的计算机已经没有能力再负荷新的操作系统了。 举例来说，Pentun-III以前的硬件配备可能已经不再适合现在的新的Linux distribution。 而且较早期的硬件配备也可能由于保存的问题或者是电子零件老化的问题， 导致这样的计算机系统反而非常容易在运行过程中出现不明的死机情况，因此在利用旧零件拼凑Linux使用的计算机系统时， 真的得要特别留意呢！

不过由于Linux运行所需要的硬件配备实在不需要太高档，因此，如果有近期汰换下来的五年内的计算机， 不必急着丢弃。由于 CPU 为 i3 等级的硬件不算太老旧，在性能方面其实也算的上非常OK了～所以， 鸟哥建议您如果有五年内的计算机被淘汰，可以拿下来测试一下，说不定能够作为你日常生活的Linux服务器， 或者是备用服务器，都是非常好用的功能哩！

但是由于不同的任务的主机所需要的硬件配备并不相同，举例来说，如果你的Linux主机是要作为企业内部的Mail server或者是Proxy server时，或者是需要使用到图形接口的运算（X Window内的Open GL等等功能）， 那么你就必须要选择高档一点的计算机配备了，使用过去的计算机零件可能并不适合呢。

下面我们稍微谈一下，如果你的Linux主要是作为小型服务器使用，并不负责学术方面的大量运算， 而且也没有使用X Window的图形接口，那你的硬件需求只要像下面这样就差不多了：

- CPU CPU只要不是老旧到会让你的硬件系统死机的都能够支持！如同前面谈到的，目前（2015）的环境中， Intel i3 系列的CPU不算太旧而且性能也不错，非常好用了。

- RAM 内存是越大越好！事实上在Linux服务器中，内存的重要性比CPU还要高的多！因为如果内存不够大， 就会使用到硬盘的内存交换空间（swap）。 而由计算机概论的内容我们知道硬盘比内存的速度要慢的多， 所以内存太小可能会影响到整体系统的性能的！尤其如果你还想要玩X window的话，那内存的容量就不能少。 对于一般的小型服务器来说，建议至少也要512MB以上的内存容量较佳。老实说，目前 DDR3 的硬件环境中， 新购系统动不动就是 4~16GB 的内存，真的是很够用了！

- Hard Disk 由于数据量与数据存取频率的不同，对于硬盘的要求也不相同。 举例来说，如果是一般小型服务器，通常重点在于容量，硬盘容量大于20GB就够用到不行了！ 但如果你的服务器是作为备份或者是小企业的文件服务器，那么你可能就得要考虑较高阶的磁盘阵列（RAID）模式了。 

  Tips 磁盘阵列（RAID）是利用硬件技术将数个硬盘整合成为一个大硬盘的方法，操作系统只会看到最后被整合起来的大硬盘。 由于磁盘阵列是由多个硬盘组成，所以可以达成速度性能、备份等任务。更多相关的磁盘阵列我们会在第十四章中介绍的。

- VGA 对于不需要X Window的服务器来说，显卡算是最不重要的一个元件了！你只要有显卡能够让计算机启动，那就够了。 但如果需要X window系统时，你的显卡最好能够拥有32MB以上的内存容量，否则跑X系统会很累喔！

- Network Interface Card 网卡是服务器上面最重要的元件之一了！目前的主板大多拥有内置10/100/1000Mbps的超高速以太网卡。 但要注意的是，不同的网卡的功能还是有点差异。举例来说，鸟哥曾经需要具有可以设置 bonding 功能的网卡， 结果，某些较低阶的 gigabit 网卡并没有办法提供这个项目的支持！眞是伤脑筋！此外，比较好的网卡通常 Linux 驱动程序也做的比较好，用起来会比较顺畅。因此，如果你的服务器是 网络 I/O 行为非常频繁的网站， 好一点的Intel/boradcom等公司的网卡应该是比较适合的喔。

- 光盘、软盘、键盘与鼠标 不要旧到你的计算机不支持就好了，因为这些配备都是非必备的喔！举例来说，鸟哥安装好Linux系统后， 可能就将该系统的光驱、鼠标、软盘机等通通拔除，只有网络线连接在计算机后面而已，其他的都是通过网络连线来管控的哩！ 因为通常服务器这东西最需要的就是稳定，而稳定的最理想状态就是平时没事不要去动他是最好的。

下面鸟哥针对一般你可能会接触到的计算机主机的用途与相关硬件配备的基本要求来说明一下好了：

- 一般小型主机且不含X Window系统：
    - 用途：家庭用NAT主机（IP分享器功能）或小型企业之非图形接口小型主机。
    - CPU：五年内出产的产品即可。
    - RAM：至少512MB，不过还是大于1GB以上比较妥当！
    - 网卡：一般的以太网卡即可应付。
    - 显卡：只要能够被Linux捉到的显卡即可，例如NVidia或ATI的主流显卡均可。
    - 硬盘：20GB以上即可！
- 桌上型（Desktop）Linux系统/含X Window：
    - 用途：Linux的练习机或办公室（Office）工作机。（一般我们会用到的环境）
    - CPU：最好等级高一点，例如 Intel I5, I7 以上等级。
    - RAM：一定要大于1GB比较好！否则容易有图形接口停顿的现象。
    - 网卡：普通的以太网卡就好了！
    - 显卡：使用256MB以上内存的显卡！（入门级的都这个容量以上了）
    - 硬盘：越大越好，最好有60GB。
- 中型以上Linux服务器：
    - 用途：中小型企业/学校单位的FTP/mail/WWW等网络服务主机。
    - CPU：最好等级高一点，例如 I5, I7 以上的多核心系统。
    - RAM：最好能够大于1GB以上，大于4GB更好！
    - 网卡：知名的broadcom或Intel等厂牌，比较稳定性能较佳！
    - 显卡：如果有使用到图形功能，则一张64MB内存的显卡是需要的！
    - 硬盘：越大越好，如果可能的话，使用磁盘阵列，或者网络硬盘等等的系统架构，能够具有更稳定安全的传输环境，更佳！
    - 建议企业用计算机不要自行组装，可购买商用服务器较佳， 因为商用服务器已经通过制造商的散热、稳定性等测试，对于企业来说，会是一个比较好的选择。

总之，鸟哥在这里仅是提出一个方向：如果你的Linux主机是小型环境使用的，实时死机也不太会影响到企业环境的运行时， 那么使用升级后被淘汰下来的零件以组成计算机系统来运行，那是非常好的回收再利用的案例。 但如果你的主机系统是非常重要的，你想要更一部更稳定的Linux服务器，那考虑系统的整体搭配与运行性能的考虑， 购买已组装测试过的商用服务器会是一个比较好的选择喔！



Tips 一般来说，目前（2015）的入门计算机机种，CPU至少都是Intel i3的2GHz系列的等级以上， 内存至少有2GB，显存也有512MB以上，所以如果您是新购置的计算机， 那么该计算机用来作为Linux的练习机，而且加装X Window系统，肯定是可以跑的吓吓叫的啦！^_^

此外，Linux开发商在释出Linux distribution之前，都会针对该版所默认可以支持的硬件做说明， 因此，你除了可以在Linux的Howto文件去查询硬件的支持度之外，也可以到各个相关的Linux distributions网站去查询呢！ 下面鸟哥列出几个常用的硬件与Linux distributions搭配的网站，建议大家想要了解你的主机支不支持该版Linux时， 务必到相关的网站去搜寻一下喔！

- Red Hat的硬件支持：https://hardware.redhat.com/?pagename=hcl
- Open SuSE的硬件支持：http://en.opensuse.org/Hardware?LANG=en_UK
- Linux对笔记本电脑的支持：http://www.linux-laptop.net/
- Linux对打印机的支持：http://www.openprinting.org/
- Linux硬件支持的中文HowTo：http://www.linux.org.tw/CLDP/HOWTO/hardware.html#hardware

总之，如果是自己维护的一个小网站，考虑到经济因素，你可以自行组装一部主机来架设。 而如果是中、大型企业，那么主机的钱不要省～因为，省了这些钱，未来主机挂点时， 光是要找出哪个元件出问题，或者是系统过热的问题，会气死人ㄟ！ 而且，要注意的就是未来你的Linux主机规划的“用途”来决定你的Linux主机硬件配备喔！相当的重要呢！

2.1.3 各硬件设备在Linux中的文件名

选择好你所需要的硬件配备后，接下来得要了解一下各硬件在Linux当中所扮演的角色啰。 这里鸟哥再次的强调一下：“在Linux系统中，每个设备都被当成一个文件来对待” 举例来说，SATA接口的硬盘的文件名称即为/dev/sd[a-d]，其中， 括号内的字母为a-d当中的任意一个，亦即有/dev/sda, /dev/sdb, /dev/sdc, 及 /dev/sdd这四个文件的意思。



Tips 这种中括号 [ ] 型式的表达式在后面的章节当中会使用得很频繁，请特别留意

另外先提出来强调一下，在Linux这个系统当中，几乎所有的硬件设备文件都在/dev这个目录内， 所以你会看到/dev/sda, /dev/sr0等等的文件名喔。

那么打印机与软盘呢？分别是/dev/lp0, /dev/fd0啰！好了，其他的周边设备呢？ 下面列出几个常见的设备与其在Linux当中的文件名啰：

设备	设备在Linux内的文件名
SCSI/SATA/USB硬盘机	/dev/sd[a-p]
USB闪存盘	/dev/sd[a-p] （与SATA相同）
VirtI/O界面	/dev/vd[a-p] （用于虚拟机内）
软盘机	/dev/fd[0-7]
打印机	/dev/lp[0-2] （25针打印机） /dev/usb/lp[0-15] （USB 接口）
鼠标	/dev/input/mouse[0-15] （通用） /dev/psaux （PS/2界面） /dev/mouse （当前鼠标）
CDROM/DVDROM	/dev/scd[0-1] （通用） /dev/sr[0-1] （通用，CentOS 较常见） /dev/cdrom （当前 CDROM）
磁带机	/dev/ht0 （IDE 界面） /dev/st0 （SATA/SCSI界面） /dev/tape （当前磁带）
IDE硬盘机	/dev/hd[a-d] （旧式系统才有）

时至今日，由于 IDE 界面的磁盘机几乎已经被淘汰，太少见了！因此现在连 IDE 界面的磁盘文件名也都被仿真成 /dev/sd[a-p] 了！此外， 如果你的机器使用的是跟网际网络供应商（ISP） 申请使用的云端机器，这时可能会得到的是虚拟机。为了加速，虚拟机内的磁盘是使用仿真器产生， 该仿真器产生的磁盘文件名为 /dev/vd[a-p] 系列的文件名喔！要注意！要注意！



Tips 更多Linux核心支持的硬件设备与文件名，可以参考如下网页：
https://www.kernel.org/doc/Documentation/devices.txt

2.1.4 使用虚拟机学习

由于近年来硬件虚拟化技术的成熟，目前普通的中阶个人计算机的 CPU 微指令集中，就已经整合了硬件虚拟化指令集了！ 所以，随便一台计算机就能够虚拟化出好几台逻辑独立的系统了！很赞！

因为虚拟化系统可以很简单的制作出相仿的硬件资源，因此我们在学习的时候，比较能够取得相同的环境来查阅学习的效果！ 所以，在本书的后续所有动作中，我们都是使用虚拟化系统来做说明！毕竟未来你实际接触到 Linux 系统时，很有可能公司交代给你的就是虚拟机了！ 趁早学也不错！

由于虚拟化的软件非常之多，网络上也有一堆朋友的教学在。如果你的系统是 windows 系列的话，鸟哥个人推荐你使用 virtualbox 这个软件！ 至于如果你原本就用 Linux 系统，例如 Fedora/Ubuntu 等系列的话，那么建议你使用原本系统内就有的虚拟机管理员来处理即可。目前 Linux 系统大多使用 KVM 这个虚拟化软件就是了。下面提供一些网站给您学习学习！鸟哥之后的章节所使用的机器，就是通过 KVM 创建出来的系统喔！ 提供给你作参考啰。

- Virtualbox 官网 （https://www.virtualbox.org）
- Virtualbox 官网教学 （https://www.virtualbox.org/manual/ch01.html）
- Fedora 教学 http://docs.fedoraproject.org/en-US/Fedora/13/html/Virtualization_Guide/part-Virtualization-Virtualization_Reference_Guide.html

2.2 磁盘分区

这一章在规划的重点是为了要安装Linux，那Linux系统是安装在计算机元件的那个部分呢？就是磁盘啦！所以我们当然要来认识一下磁盘先。 我们知道一块磁盘是可以被分区成多个分区的（partition），以旧有的Windows观点来看，你可能会有一颗磁盘并且将他分区成为C:, D:, E:盘对吧！那个C, D, E就是分区（partition）啰。但是Linux的设备都是以文件的型态存在，那分区的文件名又是什么？ 如何进行磁盘分区？磁盘分区有哪些限制？目前的 BIOS 与 UEFI 分别是啥？MSDOS 与 GPT 又是啥？ 都是我们这个小节所要探讨的内容啰。

2.2.1 磁盘连接的方式与设备文件名的关系

由第零章提到的磁盘说明，我们知道个人计算机常见的磁盘接口有两种， 分别是SATA与SAS接口，目前（2015）的主流是SATA接口。不过更老旧的计算机则有可能是已经不再流行的IDE界面喔！ 以前的IDE界面与SATA界面在Linux的磁盘代号并不相同，不过近年来为了统一处理，大部分Linux distribution已经将IDE界面的磁盘文件名也仿真成跟SATA一样了！ 所以你大概不用太担心磁盘设备文件名的问题了！

时代在改变啊～既然IDE界面都可以消失了，那磁盘文件名还有什么可谈的呢？嘿嘿！有啊！如同上一小节谈到的，虚拟化是目前很常见的一项技术， 因此你在使用的机器很可能就是虚拟机，这些虚拟机使用的“虚拟磁盘”并不是正规的磁盘界面！这种情况下面，你的磁盘文件名就不一样了！ 正常的实体机器大概使用的都是 /dev/sd[a-] 的磁盘文件名，至于虚拟机环境下面，为了加速，可能就会使用 /dev/vd[a-p] 这种设备文件名喔！ 因此在实际处理你的系统时，可能得要了解为啥会有两种不同磁盘文件名的原因才好！

例题：假设你的主机为虚拟机，里面仅有一颗VirtIO接口的磁盘，请问他在Linux操作系统里面的设备文件名为何？答：参考 2.1.3 小节的介绍，虚拟机使用 VirtIO 界面时，磁盘文件名应该是 /dev/vda 才对！

再以SATA接口来说，由于SATA/USB/SAS等磁盘接口都是使用SCSI模块来驱动的， 因此这些接口的磁盘设备文件名都是/dev/sd[a-p]的格式。 所以SATA/USB接口的磁盘根本就没有一定的顺序，那如何决定他的设备文件名呢？ 这个时候就得要根据Linux核心侦测到磁盘的顺序了！这里以下面的例子来让你了解啰。

例题：如果你的PC上面有两个SATA磁盘以及一个USB磁盘，而主板上面有六个SATA的插槽。这两个SATA磁盘分别安插在主板上的SATA1, SATA5插槽上， 请问这三个磁盘在Linux中的设备文件名为何？答：由于是使用侦测到的顺序来决定设备文件名，并非与实际插槽代号有关，因此设备的文件名如下：

1. SATA1插槽上的文件名：/dev/sda
2. SATA5插槽上的文件名：/dev/sdb
3. USB磁盘（开机完成后才被系统捉到）：/dev/sdc

通过上面的介绍后，你应该知道了在Linux系统下的各种不同接口的磁盘的设备文件名了。OK！好像没问题了吻！才不是呢～问题很大吻！ 因为如果你的磁盘被分区成两个分区，那么每个分区的设备文件名又是什么？在了解这个问题之前，我们先来复习一下磁盘的组成， 因为现今磁盘的分区与他物理的组成很有关系！

我们在计算机概论谈过磁盘的组成主要有盘片、机械手臂、磁头与主轴马达所组成， 而数据的写入其实是在盘片上面。盘片上面又可细分出扇区（Sector）与磁道（Track）两种单位，其中扇区的物理量设计有两种大小，分别是 512Bytes 与 4KBytes。假设磁盘只有一个盘片，那么盘片有点像下面这样：

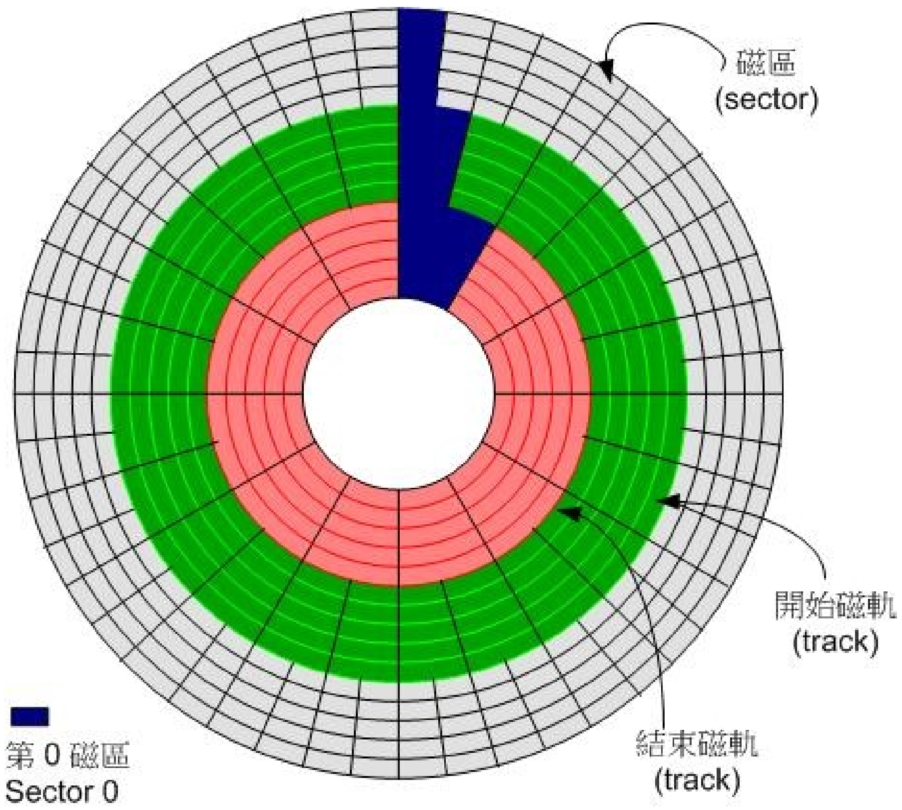


图2.2.1、盘片组成示意图

那么是否每个扇区都一样重要呢？其实整颗磁盘的第一个扇区特别的重要，因为他记录了整颗磁盘的重要信息！ 早期磁盘第一个扇区里面含有的重要信息我们称为MBR（Master Boot Record） 格式，但是由于近年来磁盘的容量不断扩大，造成读写上的一些困扰， 甚至有些大于 2TB 以上的磁盘分区已经让某些操作系统无法存取。因此后来又多了一个新的磁盘分区格式，称为 GPT（GUID partition table）！ 这两种分区格式与限制不太相同啦！

那么分区表又是啥？其实你刚刚拿到的整颗硬盘就像一根原木，你必须要在这根原木上面切割出你想要的区段， 这个区段才能够再制作成为你想要的家具！如果没有进行切割，那么原木就不能被有效的使用。 同样的道理，你必须要针对你的硬盘进行分区，这样硬盘才可以被你使用的！

2.2.2 MSDOS（MBR） 与 GPT 磁盘分区表（partition table）

但是硬盘总不能真的拿锯子来切切割割吧？那硬盘还真的是会坏掉去！那怎办？在前一小节的图示中， 我们有看到“开始与结束磁道”吧？而通常磁盘可能有多个盘片，所有盘片的同一个磁道我们称为柱面 （Cylinder）， 通常那是文件系统的最小单位，也就是分区的最小单位啦！为什么说“通常”呢？因为近来有 GPT 这个可达到 64bit 纪录功能的分区表， 现在我们甚至可以使用扇区 （sector） 号码来作为分区单位哩！厉害了！ 所以说，我们就是利用参考对照柱面或扇区号码的方式来处理啦！

也就是说，分区表其实目前有两种格式喔！我们就依序来谈谈这两种分区表格式吧。

- MSDOS （MBR） 分区表格式与限制

早期的 Linux 系统为了相容于 Windows 的磁盘，因此使用的是支持 Windows 的 MBR（Master Boot Record, 主要开机纪录区） 的方式来处理开机管理程序与分区表！而开机管理程序纪录区与分区表则通通放在磁盘的第一个扇区， 这个扇区通常是 512Bytes 的大小 （旧的磁盘扇区都是 512Bytes 喔！），所以说，第一个扇区 512Bytes 会有这两个数据：

- 主要开机记录区（Master Boot Record, MBR）：可以安装开机管理程序的地方，有446 Bytes
- 分区表（partition table）：记录整颗硬盘分区的状态，有64 Bytes

由于分区表所在区块仅有64 Bytes容量，因此最多仅能有四组记录区，每组记录区记录了该区段的启始与结束的柱面号码。 若将硬盘以长条形来看，然后将柱面以直条图来看，那么那64 Bytes的记录区段有点像下面的图示：

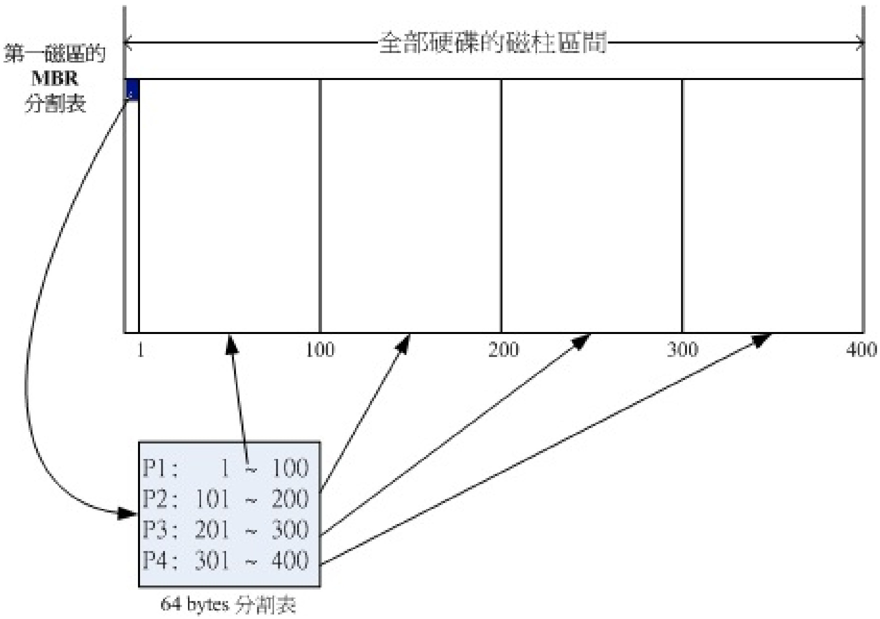


图2.2.2、磁盘分区表的作用示意图

假设上面的硬盘设备文件名为/dev/sda时，那么这四个分区在Linux系统中的设备文件名如下所示， 重点在于文件名后面会再接一个数字，这个数字与该分区所在的位置有关喔！

- P1:/dev/sda1
- P2:/dev/sda2

- P3:/dev/sda3
- P4:/dev/sda4

上图中我们假设硬盘只有400个柱面，共分区成为四个分区，第四个分区所在为第301到400号柱面的范围。 当你的操作系统为Windows时，那么第一到第四个分区的代号应该就是C, D, E, F。当你有数据要写入F盘时， 你的数据会被写入这颗磁盘的301~400号柱面之间的意思。

由于分区表就只有64 Bytes而已，最多只能容纳四笔分区的记录， 这四个分区的记录被称为主要（Primary）或延伸（Extended）分区。 根据上面的图示与说明，我们可以得到几个重点信息：

- 其实所谓的“分区”只是针对那个64 Bytes的分区表进行设置而已！
- 硬盘默认的分区表仅能写入四组分区信息
- 这四组分区信息我们称为主要（Primary）或延伸（Extended）分区
- 分区的最小单位“通常”为柱面（cylinder）
- 当系统要写入磁盘时，一定会参考磁盘分区表，才能针对某个分区进行数据的处理

咦！你会不会突然想到，为啥要分区啊？基本上你可以这样思考分区的角度：

1. 数据的安全性： 因为每个分区的数据是分开的！所以，当你需要将某个分区的数据重整时，例如你要将计算机中Windows的C 盘重新安装一次系统时， 可以将其他重要数据移动到其他分区，例如将邮件、桌面数据移动到D 盘去，那么C 盘重灌系统并不会影响到D 盘！ 所以善用分区，可以让你的数据更安全。

2. 系统的性能考虑： 由于分区将数据集中在某个柱面的区段，例如上图当中第一个分区位于柱面号码1~100号，如此一来当有数据要读取自该分区时， 磁盘只会搜寻前面1~100的柱面范围，由于数据集中了，将有助于数据读取的速度与性能！所以说，分区是很重要的！

既然分区表只有记录四组数据的空间，那么是否代表我一颗硬盘最多只能分区出四个分区？当然不是啦！有经验的朋友都知道， 你可以将一颗硬盘分区成十个以上的分区的！那又是如何达到的呢？在Windows/Linux系统中， 我们是通过刚刚谈到的延伸分区（Extended）的方式来处理的啦！延伸分区的想法是： 既然第一个扇区所在的分区表只能记录四笔数据， 那我可否利用额外的扇区来记录更多的分区信息？实际上图示有点像下面这样：

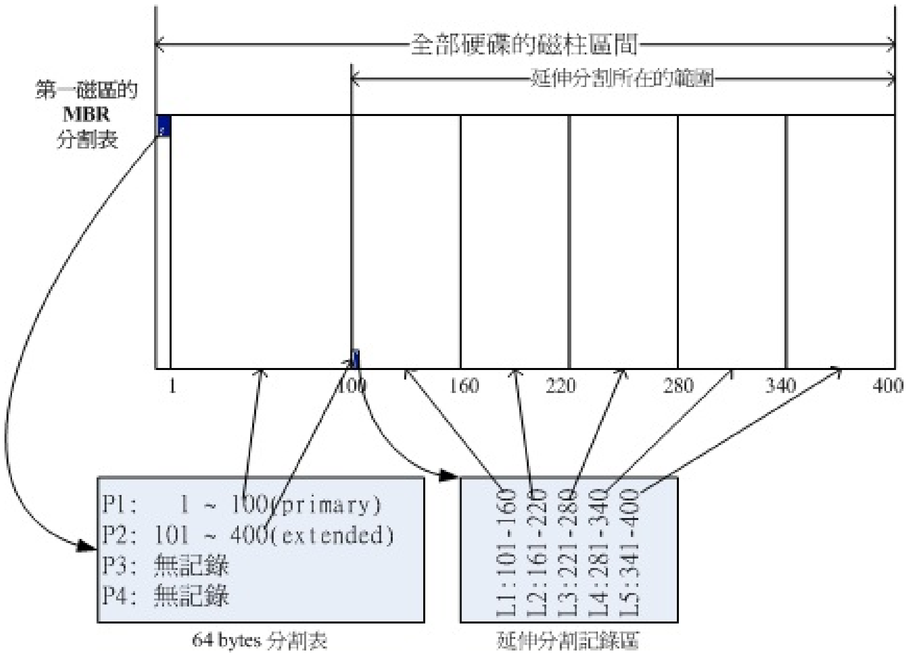




图2.2.3、磁盘分区表的作用示意图

Tips 实际上延伸分区并不是只占一个区块，而是会分佈在每个分区的最前面几个扇区来记载分区信息的！只是为了方便读者记忆，鸟哥在上图就将他简化了！有兴趣的读者可以到下面的链接瞧一瞧实际延伸分区的纪录方式：http://en.wikipedia.org/wiki/Extended_boot_record

在上图当中，我们知道硬盘的四个分区记录区仅使用到两个，P1为主要分区，而P2则为延伸分区。请注意，延伸分区的目的是使用额外的扇区来记录分区信息，延伸分区本身并不能被拿来格式化。然后我们可以通过延伸分区所指向的那个区块继续作分区的记录。

如上图右下方那个区块有继续分区出五个分区，这五个由延伸分区继续切出来的分区，就被称为逻辑分区（logical partition）。同时注意一下，由于逻辑分区是由延伸分区继续分区出来的，所以他可以使用的柱面范围就是延伸分区所设置的范围喔！也就是图中的101~400啦！

同样的，上述的分区在Linux系统中的设备文件名分别如下：

- P1:/dev/sda1
- P2:/dev/sda2
- L1:/dev/sda5
- L2:/dev/sda6
- L3:/dev/sda7
- L4:/dev/sda8
- L5:/dev/sda9

仔细看看，怎么设备文件名没有/dev/sda3与/dev/sda4呢？因为前面四个号码都是保留给Primary或Extended用的嘛！ 所以逻辑分区的设备名称号码就由5号开始了！这在 MBR 方式的分区表中是个很重要的特性，不能忘记喔！

MBR 主要分区、延伸分区与逻辑分区的特性我们作个简单的定义啰：

- 主要分区与延伸分区最多可以有四笔（硬盘的限制）
- 延伸分区最多只能有一个（操作系统的限制）
- 逻辑分区是由延伸分区持续切割出来的分区；
- 能够被格式化后，作为数据存取的分区为主要分区与逻辑分区。延伸分区无法格式化；
- 逻辑分区的数量依操作系统而不同，在Linux系统中SATA硬盘已经可以突破63个以上的分区限制；

事实上，分区是个很麻烦的东西，因为他是以柱面为单位的“连续”磁盘空间， 且延伸分区又是个类似独立的磁盘空间，所以在分区的时候得要特别注意。我们举下面的例子来解释一下好了：

例题：在Windows操作系统当中，如果你想要将D与E盘整合成为一个新的分区，而如果有两种分区的情况如下图所示， 图中的特殊颜色区块为D与E盘的示意，请问这两种方式是否均可将D与E整合成为一个新的分区？

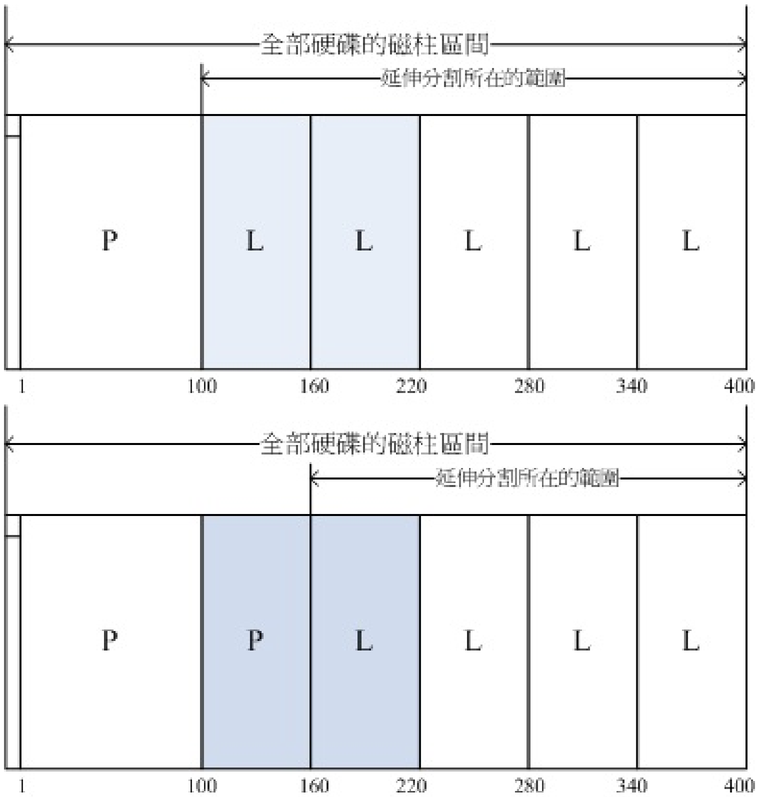


图2.2.4、磁盘空间整合示意图

答：

- 上图可以整合：因为上图的D与E同属于延伸分区内的逻辑分区，因此只要将两个分区删除，然后再重新创建一个新的分区， 就能够在不影响其他分区的情况下，将两个分区的容量整合成为一个。

- 下图不可整合：因为D与E分属主分区与逻辑分区，两者不能够整合在一起。除非将延伸分区破坏掉后再重新分区。 但如此一来会影响到所有的逻辑分区，要注意的是：如果延伸分区被破坏，所有逻辑分区将会被删除。 因为逻辑分区的信息都记录在延伸分区里面嘛！

由于第一个扇区所记录的分区表与MBR是这么的重要，几乎只要读取硬盘都会先由这个扇区先读起。 因此，如果整颗硬盘的第一个扇区（就是MBR与partition table所在的扇区）物理实体坏掉了，那这个硬盘大概就没有用了！ 因为系统如果找不到分区表，怎么知道如何读取柱面区间呢？您说是吧！下面还有一些例题您可以思考看看：

例题：如果我想将一颗大硬盘“暂时”分区成为四个partitions，同时还有其他的剩余容量可以让我在未来的时候进行规划， 我能不能分区出四个Primary？若不行，那么你建议该如何分区？答：

- 由于Primary+Extended最多只能有四个，其中Extended最多只能有一个，这个例题想要分区出四个分区且还要预留剩余容量， 因此P+P+P+P的分区方式是不适合的。因为如果使用到四个P，则即使硬盘还有剩余容量， 因为无法再继续分区，所以剩余容量就被浪费掉了。

- 假设你想要将所有的四笔记录都花光，那么P+P+P+E是比较适合的。所以可以用的四个partitions有3个主要及一个逻辑分区， 剩余的容量在延伸分区中。

- 如果你要分区超过4个以上时，一定要有Extended分区，而且必须将所有剩下的空间都分配给Extended， 然后再以logical的分区来规划Extended的空间。 另外，考虑到磁盘的连续性，一般建议将Extended的柱面号码分配在最后面的柱面内。

例题：假如我的PC有两颗SATA硬盘，我想在第二颗硬盘分区出6个可用的分区（可以被格式化来存取数据之用）， 那每个分区在Linux系统下的设备文件名为何？且分区类型各为何？至少写出两种不同的分区方式。答：由于P（primary）+E（extended）最多只能有四个，其中E最多只能有一个。现在题目要求6个可用的分区，因此不可能分出四个P。 下面我们假设两种环境，一种是将前四号全部用完，一种是仅花费一个P及一个E的情况：

- P+P+P+E的环境：

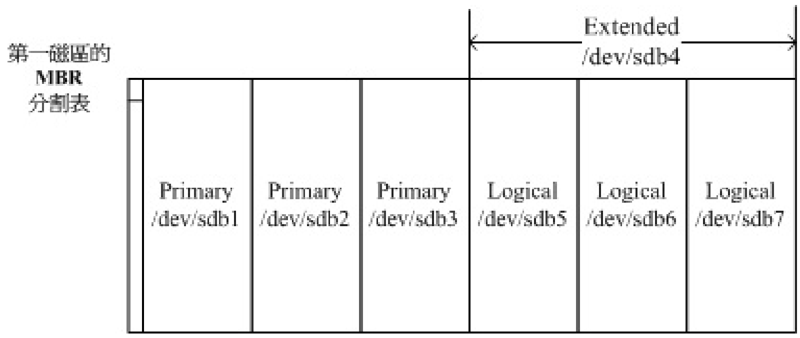
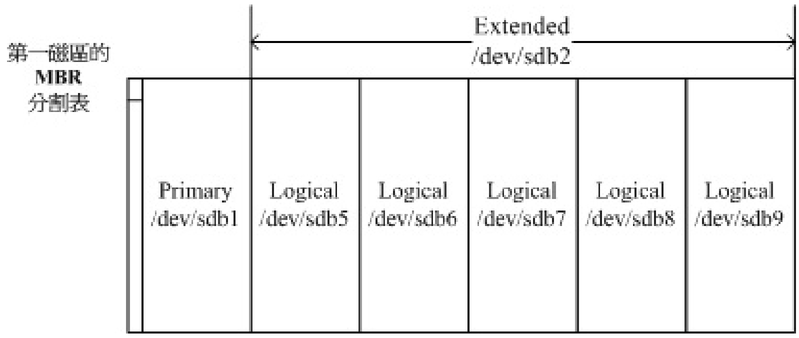


图2.2.5、分区示意图实际可用的是/dev/sdb1, /dev/sdb2, /dev/sdb3, /dev/sdb5, /dev/sdb6, /dev/sdb7这六个，至于/dev/sdb4这个延伸分区本身仅是提供来给逻辑分区创建之用。



- P+E的环境：图2.2.6、分区示意图注意到了吗？因为1~4号是保留给主要/延伸分区的，因此第一个逻辑分区一定是由5号开始的！再次强调啊！ 所以/dev/sdb3, /dev/sdb4就会被保留下来没有用到了！

MBR 分区表除了上述的主分区、延伸分区、逻辑分区需要注意之外，由于每组分区表仅有 16Bytes 而已，因此可纪录的信息真的是相当有限的！ 所以，在过去 MBR 分区表的限制中经常可以发现如下的问题：

- 操作系统无法抓取到 2.2T 以上的磁盘容量！
- MBR 仅有一个区块，若被破坏后，经常无法或很难救援。
- MBR 内的存放开机管理程序的区块仅 446Bytes，无法容纳较多的程序码。

这个 2.2TB 限制的现象在早期并不会很严重。但是，近年来硬盘厂商动不对推出的磁盘容量就高达好几个 TB 的容量！目前 （2015） 单一磁盘最大容量甚至高达 8TB 了！ 如果使用磁盘阵列的系统，像鸟哥的一组系统中，用了 24 颗 4TB 磁盘搭建出磁盘阵列，那在 Linux 下面就会看到有一颗 70TB 左右的磁盘！ 如果使用 MBR 的话...那得要 2TB/2TB 的割下去，虽然 Linux kernel 现在已经可以通过某些机制让磁盘分区高过 63 个以上，但是这样就得要割出将近 40 个分区～ 眞要命... 为了解决这个问题，所以后来就有 GPT 这个磁盘分区的格式出现了！

- GUID partition table, GPT 磁盘分区表[1]

因为过去一个扇区大小就是 512Bytes 而已，不过目前已经有 4K 的扇区设计出现！为了相容于所有的磁盘，因此在扇区的定义上面， 大多会使用所谓的逻辑区块位址（Logical Block Address, LBA）来处理。GPT 将磁盘所有区块以此 LBA（默认为 512Bytes 喔！） 来规划，而第一个 LBA 称为 LBA0 （从 0 开始编号）。

与 MBR 仅使用第一个 512Bytes 区块来纪录不同， GPT 使用了 34 个 LBA 区块来纪录分区信息！同时与过去 MBR 仅有一的区块，被干掉就死光光的情况不同， GPT 除了前面 34 个 LBA 之外，整个磁盘的最后 33 个 LBA 也拿来作为另一个备份！这样或许会比较安全些吧！ 详细的结构有点像下面的模样：

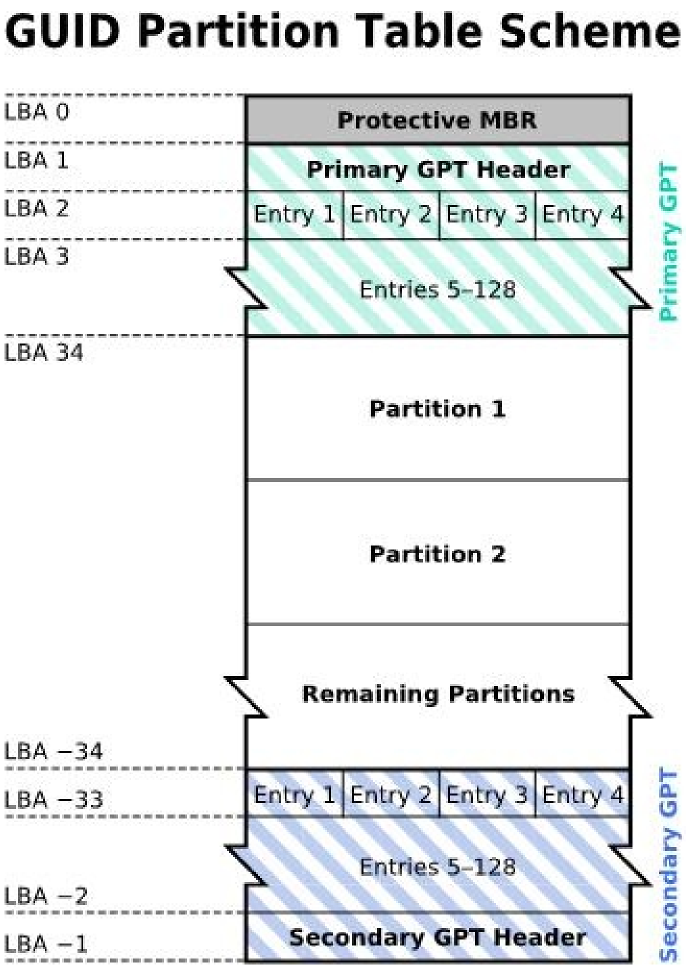


图2.2.7、GPT 分区表的结构示意图

上述图示的解释说明如下：

- LBA0 （MBR 相容区块）

  与 MBR 模式相似的，这个相容区块也分为两个部份，一个就是跟之前 446 Bytes 相似的区块，储存了第一阶段的开机管理程序！ 而在原本的分区表的纪录区内，这个相容模式仅放入一个特殊标志的分区，用来表示此磁盘为 GPT 格式之意。而不懂 GPT 分区表的磁盘管理程序， 就不会认识这颗磁盘，除非用户有特别要求要处理这颗磁盘，否则该管理软件不能修改此分区信息，进一步保护了此磁盘喔！

- LBA1 （GPT 表头纪录）

  这个部份纪录了分区表本身的位置与大小，同时纪录了备份用的 GPT 分区 （就是前面谈到的在最后 34 个 LBA 区块） 放置的位置， 同时放置了分区表的检验机制码（CRC32），操作系统可以根据这个检验码来判断 GPT 是否正确。若有错误，还可以通过这个纪录区来取得备份的 GPT（磁盘最后的那个备份区块） 来恢复 GPT 的正常运行！

- LBA2-33 （实际纪录分区信息处）

> 从 LBA2 区块开始，每个 LBA 都可以纪录 4 笔分区纪录，所以在默认的情况下，总共可以有 *432 = 128* 笔分区纪录喔！因为每个 *LBA* 有 *512Bytes*，因此每笔纪录用到 *128 Bytes* 的空间，除了每笔纪录所需要的识别码与相关的纪录之外，*GPT* 在每笔纪录中分别提供了 *64bits* 来记载开始/结束的扇区号码，因此，*GPT* 分区表对於单一分区来说，他的最大容量限制就会在“ *264* 512Bytes = 263 *1KBytes* = *233*TB = 8 ZB ”，要注意 1ZB = 230TB 啦！ 你说有没有够大了？

现在 GPT 分区默认可以提供多达 128 笔纪录，而在 Linux 本身的核心设备纪录中，针对单一磁盘来说，虽然过去最多只能到达 15 个分区，不过由于 Linux kernel 通过 udev 等方式的处理，现在 Linux 也已经没有这个限制在了！ 此外，GPT 分区已经没有所谓的主、延伸、逻辑分区的概念，既然每笔纪录都可以独立存在， 当然每个都可以视为是主分区！每一个分区都可以拿来格式化使用喔！



Tips 鸟哥一直以为核心认识的设备主要/次要号码就一定是连续的，因此一直没有注意到由于新的机制的关系，分区已经可以突破核心限制的状况！ 感谢大陆网友微博代号“学习日记博客”的提醒！此外，为了查询正确性，鸟哥还真的有注意到网络上有朋友实际拿一颗磁盘分区出 130 个以上的分区， 结果他发现 120 个以前的分区均可以格式化使用，但是 130 之后的似乎不太能够使用了！或许跟默认的 GPT 共 128 个号码有关！

虽然新版的 Linux 大多认识了 GPT 分区表，没办法，我们 server 常常需要比较大容量的磁盘嘛！不过，在磁盘管理工具上面， fdisk 这个老牌的软件并不认识 GPT 喔！要使用 GPT 的话，得要操作类似 gdisk 或者是 parted 指令才行！这部份我们会在第二篇再来谈一谈。 另外，开机管理程序方面， grub 第一版并不认识 GPT 喔！得要 grub2 以后才会认识的！开机管理程序这部份则第五篇再来谈喔！

并不是所有的操作系统都可以读取到 GPT 的磁盘分区格式喔！同时，也不是所有的硬件都可以支持 GPT 格式喔！是否能够读写 GPT 格式又与开机的检测程序有关！ 那开机的检测程序又分成啥鬼东西呢？就是 BIOS 与 UEFI 啦！那这两个又是啥东西？就让我们来聊一聊！

2.2.3 开机流程中的 BIOS 与 UEFI 开机检测程序

我们在计算机概论里面谈到了，没有执行软件的硬件是没有用的，除了会电人之外...， 而为了计算机硬件系统的资源合理分配，因此有了操作系统这个系统软件的产生。由于操作系统会控制所有的硬件并且提供核心功能， 因此我们的计算机就能够认识硬盘内的文件系统，并且进一步的读取硬盘内的软件文件与执行该软件来达成各项软件的执行目的。

问题是，你有没有发现，既然操作系统也是软件，那么我的计算机又是如何认识这个操作系统软件并且执行他的？明明开机时我的计算机还没有任何软件系统，那他要如何读取硬盘内的操作系统文件啊？嘿嘿！这就得要牵涉到计算机的开机程序了！下面就让我们来谈一谈这个开机程序吧！

基本上，目前的主机系统在载入硬件驱动方面的程序，主要有早期的 BIOS 与新的 UEFI 两种机制，我们分别来谈谈啰！

- BIOS 搭配 MBR/GPT 的开机流程

在计算机概论里面我们有谈到那个可爱的BIOS与CMOS两个东西，CMOS是记录各项硬件参数且嵌入在主板上面的储存器，BIOS则是一个写入到主板上的一个固件（再次说明，固件就是写入到硬件上的一个软件程序）。这个BIOS就是在开机的时候，计算机系统会主动执行的第一个程序了！

接下来BIOS会去分析计算机里面有哪些储存设备，我们以硬盘为例，BIOS会依据使用者的设置去取得能够开机的硬盘，并且到该硬盘里面去读取第一个扇区的MBR位置。MBR这个仅有446 Bytes的硬盘容量里面会放置最基本的开机管理程序，此时BIOS就功成圆满，而接下来就是MBR内的开机管理程序的工作了。

这个开机管理程序的目的是在载入（load）核心文件，由于开机管理程序是操作系统在安装的时候所提供的，所以他会认识硬盘内的文件系统格式，因此就能够读取核心文件，然后接下来就是核心文件的工作，开机管理程序与 BIOS 也功成圆满，将之后的工作就交给大家所知道的操作系统啦！

简单的说，整个开机流程到操作系统之前的动作应该是这样的：

1. BIOS：开机主动执行的固件，会认识第一个可开机的设备；
2. MBR：第一个可开机设备的第一个扇区内的主要开机记录区块，内含开机管理程序；
3. 开机管理程序（boot loader）：一支可读取核心文件来执行的软件；
4. 核心文件：开始操作系统的功能...

第二点要注意，如果你的分区表为 GPT 格式的话，那么 BIOS 也能够从 LBA0 的 MBR 相容区块读取第一阶段的开机管理程序码，如果你的开机管理程序能够认识 GPT 的话，那么使用 BIOS 同样可以读取到正确的操作系统核心喔！换句话说，如果开机管理程序不懂 GPT ，例如 Windows XP 的环境，那自然就无法读取核心文件，开机就失败了！



Tips 由于 LBA0 仅提供第一阶段的开机管理程序码，因此如果你使用类似 grub 的开机管理程序的话，那么就得要额外分区出一个“ BIOS boot ”的分区，这个分区才能够放置其他开机过程所需的程序码！在 CentOS 当中，这个分区通常占用 2MB 左右而已。

由上面的说明我们会知道，BIOS与MBR都是硬件本身会支持的功能，至于Boot loader则是操作系统安装在MBR上面的一套软件了。由于MBR仅有446 Bytes而已，因此这个开机管理程序是非常小而美的。 这个boot loader的主要任务有下面这些项目：

- 提供菜单：使用者可以选择不同的开机项目，这也是多重开机的重要功能！
- 载入核心文件：直接指向可开机的程序区段来开始操作系统；
- 转交其他loader：将开机管理功能转交给其他loader负责。

上面前两点还容易理解，但是第三点很有趣喔！那表示你的计算机系统里面可能具有两个以上的开机管理程序呢！ 有可能吗？我们的硬盘不是只有一个MBR而已？是没错啦！但是开机管理程序除了可以安装在MBR之外， 还可以安装在每个分区的开机扇区（boot sector）喔！瞎密？分区还有各别的开机扇区喔？ 没错啊！这个特色才能造就“多重开机”的功能啊！

我们举一个例子来说，假设你的个人计算机只有一个硬盘，里面切成四个分区，其中第一、二分区分别安装了Windows及Linux， 你要如何在开机的时候选择用Windows还是Linux开机呢？假设MBR内安装的是可同时认识Windows/Linux操作系统的开机管理程序， 那么整个流程可以图示如下：

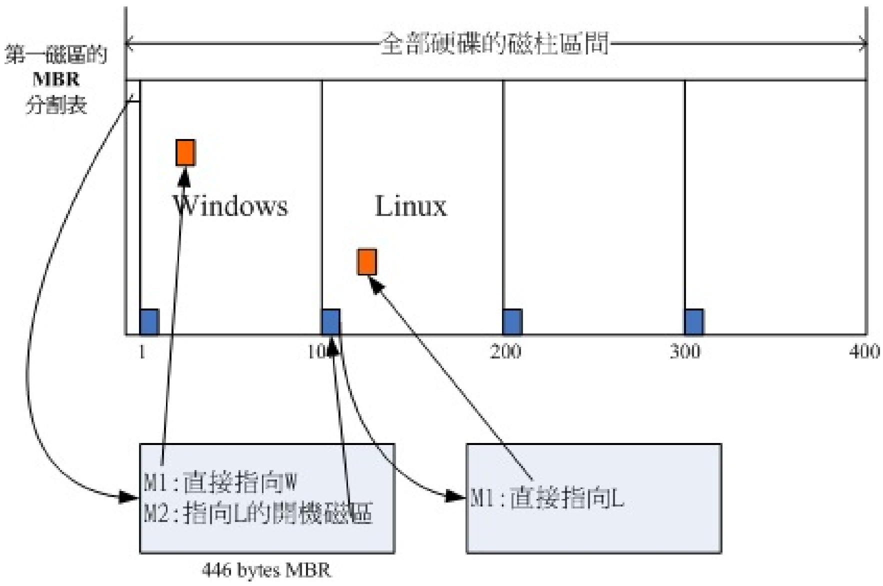


图2.2.8、开机管理程序的工作执行示意图

在上图中我们可以发现，MBR的开机管理程序提供两个菜单，菜单一（M1）可以直接载入Windows的核心文件来开机； 菜单二（M2）则是将开机管理工作交给第二个分区的开机扇区（boot sector）。当使用者在开机的时候选择菜单二时， 那么整个开机管理工作就会交给第二分区的开机管理程序了。 当第二个开机管理程序启动后，该开机管理程序内（上图中）仅有一个开机菜单，因此就能够使用Linux的核心文件来开机啰。 这就是多重开机的工作情况啦！我们将上图作个总结：

- 每个分区都拥有自己的开机扇区（boot sector）
- 图中的系统盘为第一及第二分区，
- 实际可开机的核心文件是放置到各分区内的！
- loader只会认识自己的系统盘内的可开机核心文件，以及其他loader而已；

- loader可直接指向或者是间接将管理权转交给另一个管理程序。

那现在请你想一想，为什么人家常常说：“如果要安装多重开机， 最好先安装Windows再安装Linux”呢？这是因为：

- Linux在安装的时候，你可以选择将开机管理程序安装在MBR或各别分区的开机扇区， 而且Linux的loader可以手动设置菜单（就是上图的M1, M2...），所以你可以在Linux的boot loader里面加入Windows开机的选项；
- Windows在安装的时候，他的安装程序会主动的覆盖掉MBR以及自己所在分区的开机扇区，你没有选择的机会， 而且他没有让我们自己选择菜单的功能。

因此，如果先安装Linux再安装Windows的话，那MBR的开机管理程序就只会有Windows的项目，而不会有Linux的项目 （因为原本在MBR内的Linux的开机管理程序就会被覆盖掉）。 那需要重新安装Linux一次吗？当然不需要，你只要用尽各种方法来处理MBR的内容即可。 例如利用Linux的救援模式来挽救MBR啊！



Tips 开机管理程序与Boot sector的观念是非常重要的，我们会在第十九章分别介绍，您在这里只要先对于（1）开机需要开机管理程序， 而（2）开机管理程序可以安装在MBR及Boot Sector两处这两个观念有基本的认识即可， 一开始就背太多东西会很混乱啦！

- UEFI BIOS 搭配 GPT 开机的流程 [2]

我们现在知道 GPT 可以提供到 64bit 的寻址，然后也能够使用较大的区块来处理开机管理程序。但是 BIOS 其实不懂 GPT 耶！还得要通过 GPT 提供相容模式才能够读写这个磁盘设备～而且 BIOS 仅为 16 位的程序，在与现阶段新的操作系统接轨方面有点弱掉了！ 为了解决这个问题，因此就有了 UEFI （Unified Extensible Firmware Interface） 这个统一可延伸固件界面的产生。

UEFI 主要是想要取代 BIOS 这个固件界面，因此我们也称 UEFI 为 UEFI BIOS 就是了。 UEFI 使用 C 程序语言，比起使用组合语言的传统 BIOS 要更容易开发！也因为使用 C 语言来撰写，因此如果开发者够厉害，甚至可以在 UEFI 开机阶段就让该系统了解 TCP/IP 而直接上网！ 根本不需要进入操作系统耶！这让小型系统的开发充满各式各样的可能性！

基本上，传统 BIOS 与 UEFI 的差异可以用T客帮杂志汇整的表格来说明：

比较项目	传统 **BIOS**	**UEFI**
使用程序语言	组合语言	C 语言
硬件资源控制	使用中断 （IRQ） 管理 不可变的内存存取 不可变得输入/输出存取	使用驱动程序与协定
处理器运行环境	16 位	CPU 保护模式
扩充方式	通过 IRQ 链接	直接载入驱动程序
第三方厂商支持	较差	较佳且可支持多平台
图形化能力	较差	较佳
内置简化操作系统前环境	不支持	支持

从上头我们可以发现，与传统的 BIOS 不同，UEFI 简直就像是一个低阶的操作系统～甚至于连主板上面的硬件资源的管理， 也跟操作系统相当类似，只需要载入驱动程序即可控制操作。同时由于程控得宜，一般来说，使用 UEFI 接口的主机，在开机的速度上要比 BIOS 来的快上许多！ 因此很多人都觉得 UEFI 似乎可以发展成为一个很有用的操作系统耶～不过，关于这个，你无须担心未来除了 Linux 之外，还得要增加学一个 UEFI 的操作系统啦！为啥呢？

UEFI 当初在发展的时候，就制定一些控制在里头，包括硬件资源的管理使用轮询 （polling） 的方式来管理，与 BIOS 直接了解 CPU 以中断的方式来管理比较， 这种 polling 的效率是稍微慢一些的，另外，UEFI 并不能提供完整的高速缓存功能，因此执行效率也没有办法提升。不过由于载入所有的 UEFI 驱动程序之后， 系统会打开一个类似操作系统的 shell 环境，使用者可以此环境中执行任意的 UEFI 应用程序，而且效果比 MSDOS 更好哩。

所以啰，因为效果华丽但性能不佳，因此这个 UEFI 大多用来作为启动操作系统之前的硬件检测、开机管理、软件设置等目的，基本上是比较难的。 同时，当载入操作系统后，一般来说，UEFI 就会停止工作，并将系统交给操作系统，这与早期的 BIOS 差异不大。比较特别的是，某些特定的环境下， 这些 UEFI 程序是可以部份继续执行的，以协助某些操作系统无法找到特定设备时，该设备还是可以持续运行。

此外，由于过去 cracker 经常借由 BIOS 开机阶段来破坏系统，并取得系统的控制权，因此 UEFI 加入了一个所谓的安全启动 （secure boot） 机制， 这个机制代表着即将开机的操作系统必须要被 UEFI 所验证，否则就无法顺利开机！微软用了很多这样的机制来管理硬件。 不过加入这个机制后，许多的操作系统，包括 Linux ，就很有可能无法顺利开机喔！所以，某些时刻，你可能得要将 UEFI 的 secure boot 功能关闭， 才能够顺利的进入 Linux 哩！ （这一点让自由软件工作者相当感冒啦！）

另外，与 BIOS 模式相比，虽然 UEFI 可以直接取得 GPT 的分区表，不过最好依旧拥有 BIOS boot 的分区支持， 同时，为了与 windows 相容，并且提供其他第三方厂商所使用的 UEFI 应用程序储存的空间，你必须要格式化一个 vfat 的文件系统， 大约提供 512MB 到 1G

左右的容量，以让其他 UEFI 执行较为方便。



Tips 由于 UEFI 已经克服了 BIOS 的 1024 柱面的问题，因此你的开机管理程序与核心可以放置在磁盘开始的前 2TB 位置内即可！加上之前提到的 BIOS boot 以及 UEFI 支持的分区，基本上你的 /boot 目录几乎都是 /dev/sda3 之后的号码了！这样开机还是没有问题的！所以要注意喔！与以前熟悉的分区状况已经不同， /boot 不再是 /dev/sda1 啰！很有趣吧！

2.2.4 Linux安装模式下，磁盘分区的选择（极重要）

在 windows 系统重灌之前，你可能都会事先考虑，到底系统盘 C 盘要有多少容量？而数据碟 D 盘又要给多大容量等等， 然后实际安装的时候，你会发现到其实 C 盘之前会有个 100MB 的分区被独立出来～所以实际上你就会有三个分区就是了。那 Linux 下面又该如何设计类似的东西呢？

- 目录树结构 （directory tree）

我们前面有谈过Linux内的所有数据都是以文件的形态来呈现的，所以啰，整个Linux系统最重要的地方就是在于目录树架构。 所谓的目录树架构（directory tree）就是以根目录为主，然后向下呈现分支状的目录结构的一种文件架构。 所以，整个目录树架构最重要的就是那个根目录（root directory），这个根目录的表示方法为一条斜线“/”， 所有的文件都与目录树有关。目录树的呈现方式如下图所示：

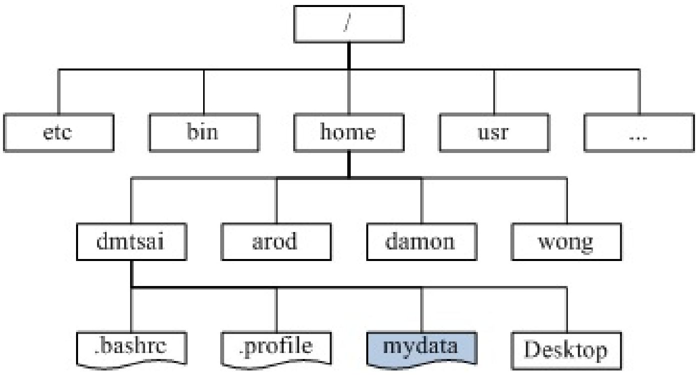


图2.2.9、目录树相关性示意图

如上图所示，所有的文件都是由根目录（/）衍生来的，而次目录之下还能够有其他的数据存在。上图中长方形为目录， 波浪形则为文件。那当我们想要取得mydata那个文件时，系统就得由根目录开始找，然后找到home接下来找到dmtsai， 最终的文件名为：/home/dmtsai/mydata的意思。

我们现在知道整个Linux系统使用的是目录树架构，但是我们的文件数据其实是放置在磁盘分区当中的， 现在的问题是“如何结合目录树的架构与磁盘内的数据”呢？ 这个时候就牵扯到“挂载（mount）”的问题啦！

- 文件系统与目录树的关系（挂载）

所谓的“挂载”就是利用一个目录当成进入点，将磁盘分区的数据放置在该目录下； 也就是说，进入该目录就可以读取该分区的意思。这个动作我们称为“挂载”，那个进入点的目录我们称为“挂载点”。 由于整个Linux系统最重要的是根目录，因此根目录一定需要挂载到某个分区的。 至于其他的目录则可依使用者自己的需求来给予挂载到不同的分区。我们以下图来作为一个说明：

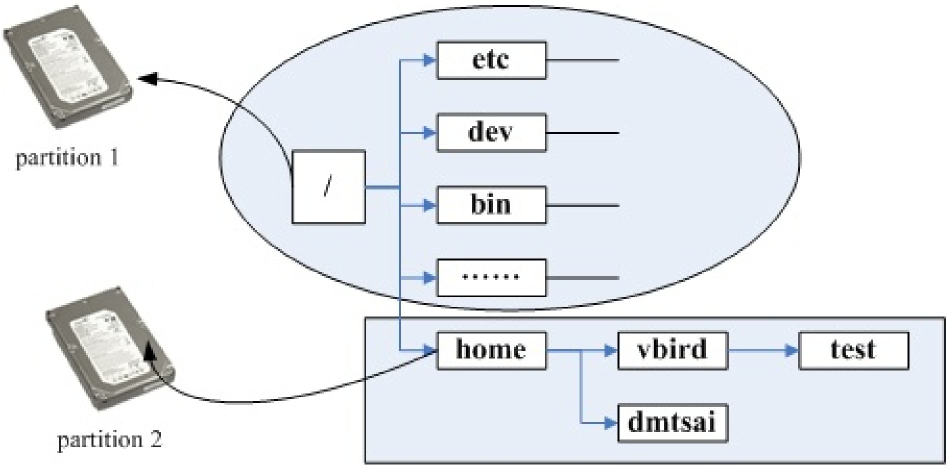


图2.2.10、目录树与分区之间的相关性

上图中假设我的硬盘分为两个分区，partition 1是挂载到根目录，至于partition 2则是挂载到/home这个目录。 这也就是说，当我的数据放置在/home内的各次目录时，数据是放置到partition 2的，如果不是放在/home下面的目录， 那么数据就会被放置到partition 1了！



Tips windows 也是用挂载的观念啊！鸟哥上课经常谈到的范例就是，当你拿 USB 磁盘放置到你的 windows 时，系统会侦测到一个 F 盘好了， 那你想要读取 USB 的数据，要去哪里啊？当然就去 F 啰！同样的这颗 USB，当你拿到学校的 windows 时，却显示的是 H 盘好了， 那你要读取 USB 的数据还是去 F 盘吗？当然不是，你会去 H 盘啊！这个“设备与磁盘分区对应的关系，就是 windows 概念下的挂载”啦！ 这样说，有没有比较好理解？

其实判断某个文件在那个partition下面是很简单的，通过反向追踪即可。以上图来说， 当我想要知道/home/vbird/test这个文件在哪个partition时，由test --> vbird --> home --> /，看那个“进入点”先被查到那就是使用的进入点了。 所以test使用的是/home这个进入点而不是/喔！

例题：现在让我们来想一想，我的计算机系统如何读取光盘内的数据呢？在Windows里面使用的是“光驱”的代号方式处理（假设为E盘时）， 但在Linux下面我们依旧使用目录树喔！在默认的情况下，Linux是将光驱的数据放置到/media/cdrom里头去的。 如果光盘片里面有个文

件文件名为“我的文件”时，那么这个文件是在哪里？答：这个文件最终会在如下的完整文件名中：

- Windows： 桌面\我的计算机\E:\我的文件
- Linux： /media/cdrom/我的文件

如果光驱并非被挂载到/media/cdrom，而是挂载到/mnt这个目录时，刚刚读取的这个文件的文件名会变成：

- /mnt/我的文件

如果你了解这个文件名，这表示你已经知道挂载的意义了！初次接触Linux时，这里最容易搞混，因为他与Windows的分区代号完全不一样！

- distributions安装时，挂载点与磁盘分区的规划：

既然我们在Linux系统下使用的是目录树系统，所以安装的时候自然就得要规划磁盘分区与目录树的挂载了。 实际上，在Linux安装的时候已经提供了相当多的默认模式让你选择分区的方式了， 不过，无论如何，分区的结果可能都不是很能符合自己主机的样子！因为毕竟每个人的“想法”都不太一样！ 因此，强烈建议使用“自订安装, Custom ”这个安装模式！在某些Linux distribution中，会将这个模式写的很厉害，叫做是“Expert, 专家模式”，这个就厉害了， 请相信您自己，了解上面的说明后，就请自称为专家了吧！没有问题！

- 自订安装“Custom”：
    - A：初次接触Linux：只要分区“ / ”及“swap”即可：

      通常初次安装Linux系统的朋友们，我们都会建议他直接以一个最大的分区“ / ”来安装系统。 这样作有个好处，就是不怕分区错误造成无法安装的困境！例如/usr是Linux的可执行程序及相关的文件摆放的目录， 所以他的容量需求蛮大的，万一你分区了一块分区给/usr，但是却给的不够大，那么就伤脑筋了！ 因为会造成无法将数据完全写入的问题，就有可能会无法安装啦！因此如果你是初次安装的话， 那么可以仅分区成两个分区“ / 与 Swap ”即可。

    - B：建议分区的方法：预留一个备用的剩余磁盘容量！

      在想要学习Linux的朋友中，最麻烦的可能就是得要常常处理分区的问题，因为分区是系统管理员很重要的一个任务。 但如果你将整个硬盘的容量都用光了，那么你要如何练习分区呢？^_^。所以鸟哥在后续的练习中也会这样做， 就是请你特别预留一块不分区的磁盘容量，作为后续练习时可以用来分区之用！

      此外，预留的分区也可以拿来做为备份之用。因为我们在实际操作Linux系统的过程中， 可能会发现某些script或者是重要的文件很值得备份时，就可以使用这个剩余的容量分区出新的分区， 并使用来备份重要的配置文件或者是script。这有个最大的好

处， 就是当我的Linux重新安装的时候，我的一些软件或工具程序马上就可以直接在硬盘当中找到！呵呵！重新安装比较便利啦。 为什么要重新安装？因为没有安装过Linux十次以上，不要说你学会了Linux了啦！慢慢体会这句话吧！ ^_^

- 选择Linux安装程序提供的默认硬盘分区方式：

  对于首次接触Linux的朋友们，鸟哥通常不建议使用各个distribution所提供默认的Server安装方式， 因为会让你无法得知Linux在搞什么鬼，而且也不见得可以符合你的需求！而且要注意的是， 选择Server的时候，请“确定”你的硬盘数据是不再需要！因为Linux会自动的把你的硬盘里面旧有的数据全部杀掉！

现在你知道Linux为什么不好学了吧？因为很多基础知识都得要先了解！否则连安装都不知道怎么安装～ 现在你知道Linux的可爱了吧！因为如果你学会了，嘿嘿！很多计算机系统/操作系统的概念都很清晰， 转换到不同的信息跑道是比较容易的喔！^_^

2.3 安装Linux前的规划

安装最重要的第一件事，就是要取得Linux distributions的光盘数据，该如何去下载？ 目前有这么多的distributions，你应该要选择哪一个版本比较好？为什么会比较好？ 在台湾，你可以在哪里下载你所需要的Linux distribution呢？这是这一小节所要讨论的喔！

2.3.1 选择适当的distribution

就如同第一章、Linux是什么里面的distributions谈到的， 事实上每个Linux distributions使用的都是来自于http://www.kernel.org官方网站所提供的Linux核心，各家distribution使用的软件其实也都是大同小异， 最大的差别或许就是在于软件的安装模式而已。所以，您只要选择其中一套，并且玩得出神入化， 那么Linux肯定可以学的成的。

不过，由于近年来网络环境实在不很安全，因此你在选择distribution时，特别要了解到该distribution适合的环境， 并且最好选择最新的distribution较佳喔！ 以鸟哥来说，如果是将Linux定位在服务器上面的话，那么Red Hat Enterprise Linux及SuSE Enterprise Linux应该是很不错的选择，因为他的版本更动幅度较小，并且更新支持的期限较长的原因。

在我们这次的练习中，不想给大家太沉重的$$负担啦，所以鸟哥选择CentOS这一个号称与RHEL完全相容的版本来练习， 目前（2015/05）最新的版本是CentOS 7.1版。不过，从CentOS 7.0 版本开始，安装光盘已经不再提供 386 相容版本了， 亦即仅有 64 位的硬件才能够使用该安装光盘来装系统了！旧的 32 位硬件系统已经不主动提供安装光盘了喔！

你可以选择到CentOS的官方网站去下载最新的版本，不过我们在台湾嘛！台湾有映设站台（mirror site）， 所以由映设站台来下载比较快啊！下面列出CentOS的下载点：

- 国家高速网络中心：http://ftp.twaren.net/Linux/CentOS/7/isos/
- 昆山科技大学：http://ftp.ksu.edu.tw/FTP/Linux/CentOS/7/isos/
- CentOS官方网站：http://mirror.centos.org/centos/7/isos/

CentOS 7.x 有提供完整版本 （everything） 以及大部分安装软件的 DVD1 版本，鸟哥建议如果你的网络速度够大，下载 everything 版本即可， 如果你得要使用光驱来安装的话，那直接下载 DVD 版本并且烧录到 DVD 光盘上面即可安装了。如果不想要安装，只想要看看到底开机会是什么 Linux 环境， 可以下载 LiveCD/LiveGNOME/LiveKDE 等版本来测试喔！如果想要练功，可以直接使用最小安装光盘版 （Minimal） 来安装！

不知道你有没有发现，怎么我想要下载的文件名会是 CentOS-7-x86_64-Everything-1503-01.iso 这样的格式？那个 1503 是啥东西啊？ 其实从 CentOS 7 之后，版本命名的依据就跟发表的日期有关了！那个 CentOS-7 讲的是 7.x 版本，x86_64 指的是 64 位操作系统，Everything 指的是包山包海的版本，1503 指的是 2015 年的 3 月发表的版本， 01.iso 则得要与 CentOS7 搭配，所以是 CentOS 7.1 版的意思！ 这样有看懂吗？



Tips 你所下载的文件扩展名是.iso，这就是所谓的image文件（镜像文件）。这种image文件是由光盘直接烧录成文件的， 文件非常的大，建议你不要使用浏览器（IE/Firefox..）来下载，可以使用FTP用户端程序来下载，例如Filezilla （http://filezilla-project.org/download.php）等。这样比较不需要担心断线的问题，因为可以续传啊！

此外，这种镜像文件可不能以数据格式烧录成为光盘/DVD的！你必须要使用烧录程序的功能， 将他<u>以“镜像文件格式”烧录成为光盘或DVD才行</u>！切记不要使用烧录数据文件格式来烧录喔！重要重要！

2.3.2 主机的服务规划与硬件的关系

我们前面已经提过，由于主机的服务目的不同，所需要的硬件等级与配备自然也就不一样！下面鸟哥稍微提一提每种服务可能会需要的硬件配备规划，当然，还是得提醒， 每个朋友的需求都不一样，所以设计您的主机之前，请先针对自己的需求进行考虑。而， 如果您不知道自己的考虑为何，那么就先拿一部普通的计算机来玩一玩吧！不过要记得！ 不要将重要数据放在练习用的Linux主机上面。

- 打造Windows与Linux共存的环境：

在某些情况之下，你可能会想要在“一部主机上面安装两套以上的操作系统”， 例如下面这些状况：

- 我的环境里面仅能允许我拥有一部主机，不论是经济问题还是空间问题～
- 因为目前各主要硬件还是针对Windows进行驱动程序的开发，我想要同时保有Windows操作系统与Linux操作系统， 以确定在Linux下面的硬件应该使用那个I/O port或者是IRQ的分配等等；
- 我的工作需要同时使用到Windows与Linux操作系统。

果真如此的话，那么刚刚我们在上一个小节谈到的开机流程与多重开机的数据就很重要了。因为需要如此你才能够在一部主机上面操弄两种不同的操作系统嘛！

如果你的Linux主机已经是想要拿来作为某些服务之用时，那么务必不要选择太老旧的硬件喔！ 前面谈到过，太老旧的硬件可能会有电子零件老化的问题～另外，如果你的Linux主机必须要全年无休的开机着， 那么摆放这部主机的位置也需要选择啊！好了，下面再来谈一谈，在一般小型企业或学校单位中， 常见的某些服务与你的硬件关系有哪些？

- NAT（达成IP分享器的功能）：

通常小型企业或者是学校单位大多仅会有一条对外的连线，然后全公司/学校内的计算机全部通过这条连线连到网际网络上。 此时我们就得要使用IP分享器来让这一条对外连线分享给所有的公司内部员工使用。 那么Linux能不能达到此一IP分享的功能呢？当然可以，就是通过NAT服务即可达成这项任务了！

在这种环境中，由于Linux作为一个内/外分离的实体，因此网络流量会比较大一点。 此时Linux主机的网卡就需要比较好些的配备。其他的CPU、RAM、硬盘等等的影响就小很多。 事实上，单利用Linux作为NAT主机来分享IP是很不智的～因为PC的耗电能力比IP分享器要大的多～

那么为什么你还要使用Linux作为NAT呢？因为Linux NAT还可以额外的加装很多分析软件，可以用来分析用户端的连线，或者是用来控制带宽与流量，达到更公平的带宽使用呢！ 更多的功能则有待后续更多的学习啰！你也可以参考我们在服务器架设篇当中的数据啰！

- SAMBA（加入Windows网络上的芳邻）：

在你的Windows系统之间如何传输数据呢？当然就是通过网络上的芳邻来传输啦！那还用问。 这也是学校老师在上课过程中要分享数据给同学常用的机制了。问题是，Windows 7 的网芳一般只能同时分享十部用户端连线，超过的话就得要等待了～真不人性化。

我们可以使用Linux上面的SAMBA这个软件来达成加入Windows网芳的功能喔！SAMBA的性能不错， 也没有用户端连线数的限制，相当适合于一般学校环境的文件服务器（file server）的角色呢！

这种服务器由于分享的数据量较大，对于系统的网卡与硬盘的大小及速度就比较重要， 如果你还针对不同的使用者提供文件服务器功能，那么/home这个目录可以考虑独立出来，并且加大容量。

- Mail（邮件服务器）：

邮件服务器是非常重要的，尤其对于现代人来说，电子邮件几乎已经取代了传统的人工邮件递送了。 拜硬盘价格大跌及Google/Yahoo/MicroSoft公平竞争之赐，一般免费的email信箱几乎都提供了很不错的邮件服务， 包过Web接口的传输、大于2GB以上的容量空间及全年无休的服务等等。例如非常多人使用的gmail就是一例： http://gmail.com。

虽然免费的信箱已经非常够用了，老实说，鸟哥也不建议您架设mail server了。问题是， 如果你是一间私人单位的公司，你的公司内传送的email是具有商业机密或隐私性的，那你还想要交给免费信箱去管理吗？ 此时才有需要架设mail server啰。在mail server上面，重要的也是硬盘容量与网卡速度，在此情境中，也可以将/var目录独立出来，并加大容量。

- Web（WWW服务器）：

WWW服务器几乎是所有的网络主机都会安装的一个功能，因为他除了可以提供Internet的WWW连线之外， 很多在网络主机上面的软件功能（例如某些分析软件所提供的最终分析结果的画面）也都使用WWW作为显示的接口， 所以这家伙真是重要到不行的。

CentOS使用的是Apache这套软件来达成WWW网站的功能，在WWW服务器上面，如果你还有提供数据库系统的话， 那么CPU的等级就不能太低，而最重要的则是RAM了！要增加WWW服务器的性能，通常提升RAM是一个不错的考虑。

- DHCP（提供用户端自动取得IP的功能）：

如果你是个区域网络管理员，你的区网内共有20部以上的计算机给一般员工使用，这些员工假设并没有计算机网络的维护技能。 那你想要让这些计算机在连上Internet时需要手动去设置IP还是他可以自动的取得IP呢？当然是自动取得比较方便啦！ 这就是DHCP服务的功能了！用户端计算机只要选择“自动取得IP”，其他的，就是你系统管理员在DHCP服务器上面设置一下即可。 这个咚咚的硬件要求可以不必很高啰。

- FTP：

常常看到很多朋友喜欢架设FTP去进行网络数据的传输，甚至很多人会架设地下FTP网站去传输些违法的数据。 老实说，“FTP传输再怎么地下化也是很容易被捉到的”啦！所以，鸟哥相当不建议您架设FTP的喔！ 不过，对于大专院校来说，因为常常需要分享给全校师生一些免费的资源， 此时匿名使用者的FTP软件功能就很需要存在了。对于FTP的硬件需求来说，硬盘容量与网卡好坏相关性较高。

大致上我们会安装的服务器软件就是这一些啰！ 当然啦，还是那句老话，在目前你刚接触Linux的这个阶段中，还是以Linux基础为主， 鸟哥也希望你先了解Linux的相关主机操作技巧，其他的建站，未来再谈吧！ 而上面列出的各项服务，仅是提供给你，如果想要架设某种网络服务的主机时，你应该如何规划主机比较好！

2.3.3 主机硬盘的主要规划

系统对于硬盘的需求跟刚刚提到的主机开放的服务有关，那么除了这点之外，还有没有其他的注意事项呢？ 当然有，那就是数据的分类与数据安全性的考虑。所谓的“数据安全”并不是指数据被网络cracker所破坏， 而是指“当主机系统的硬件出现问题时，你的文件数据能否安全的保存”之意。

常常会发现网络上有些朋友在问“我的Linux主机因为跳电的关系，造成不正常的关机，结果导致无法开机，这该如何是好？” 呵呵，幸运一点的可以使用fsck来解决硬盘的问题，麻烦一点的可能还需要重新安装Linux呢！伤脑筋吧！另外， 由于Linux是多用户多任务的环境，因此很可能上面已经有很多人的数据在其中了， 如果需要重新安装的话，光是搬移与备份数据就会疯掉了！所以硬盘的分区考虑是相当重要的！

虽然我们在本章的第二小节部分有谈论过磁盘分区了，但是，硬盘的规划对于Linux新鲜人而言，那将是造成你“头疼”的主要凶手之一！ 因为硬盘的分区技巧需要对于Linux文件结构有相当程度的认知之后才能够做比较完善的规划的！ 所以，在这里你只要有个基础的认识即可。老实说，没有安装过十次以上的Linux系统，是学不会Linux与磁盘分区的啦！

无论如何，下面还是说明一下基本硬盘分区的模式吧！

- 最简单的分区方法： 这个在上面第二节已经谈过了，就是仅分区出根目录与内存交换空间（ / & swap ）即可。 然后再预留一些剩余的磁盘以供后续的练习之用。不过，这当然是不保险的分区方法（所以鸟哥常常说这是“懒人分区法”）！ 因为如果任何一个小细节坏掉（例如坏轨的产生），你的根目录将可能整个的损毁～挽救方面较困难！
- 稍微麻烦一点的方式： 较麻烦一点的分区方式就是先分析这部主机的未来用途，然后根据用途去分析需要较大容量的目录， 以及读写较为频繁的目录，将这些重要的目录分别独立出来而不与根目录放在一起， 那当这些读写较频繁的磁盘分区有问题时，至少不会影响到根目录的系统数据，那挽救方面就比较容易啊！ 在默认的CentOS环境中，下面的目录是比较符合容量大且（或）读写频繁的目录啰：
    - /boot
    - /
    - /home
    - /var
    - Swap

以鸟哥为例，通常我会希望我的邮件主机大一些，因此我的/var通常会给个数GB的大小， 如此一来就可以不担心会有邮件空间不足的情况了！另外，由于我开放SAMBA服务， 因此提供每个研究室内人员的数据备份空间，所以啰，/home所开放的空间也很大！至于/usr/的容量，大概只要给2-5GB即可！凡此种种均与您当初预计的主机服务有关！ 因此，请特别注意您的服务项目！然后才来进行硬盘的规划。

2.3.4 鸟哥的两个实际案例

这里说一下鸟哥的两个实际的案例，这两个案例是目前还在运行的主机喔！ 要先声明的是，鸟哥的范例不见得是最好的，因为每个人的考虑并不一样。我只是提供相对可以使用的方案而已喔！

- 案例一：家用的小型Linux服务器，IP分享与文件分享中心：
- 提供服务： 提供家里的多部计算机的网络连线分享，所以需要NAT功能。提供家庭成员的数据存放容量，由于家里使用Windows系统的成员不少， 所以创建SAMBA服务器，提供网芳的网络磁盘功能。
- 主机硬件配备：
    - CPU使用 AMD Athlon 4850e 省电型 CPU
    - 内存大小为 4GB
    - 两张网卡，控制芯片为常见的螃蟹卡（Realtek）
    - 只有一颗 640GB 的磁盘
    - 显卡为 CPU 内的内置显卡 （Radeon HD 3200）
    - 安装完毕后将屏幕,键盘,鼠标,DVD-ROM等配备均移除，仅剩下网络线与电源线。
- 硬盘分区：

    - 分成 /, /usr, /var, /tmp等目录均独立；
    - 1 GB的Swap；
    - 安装比较过时的 CentOS 5.x 最新版
- 案例二：提供Linux的PC丛集（Cluster）计算机群：
- 提供服务： 提供研究室成员对于模式仿真的软、硬件平台，主要提供的服务并非网际网络服务，而是研究室内部的研究工作分析。
- 主机硬件配备：
    - 利用两部多核系统处理器 （一部 20核 40绪，一部 12核 24绪），搭配 10G 网卡组合而成
    - 使用内置的显卡
    - 运算用主机仅一颗磁盘，储存用主机提供 8 颗 2TB 磁盘组成的磁盘阵列
    - 一部 128GB 内存，一部 96GB 内存
- 硬盘分区：
    - 运算主机方面，整颗磁盘仅分 /boot, / 及 swap 而已
    - 储存主机方面，磁盘阵列分成两颗磁盘，一颗 100G 给系统用，一颗 12T 给数据用。系统磁盘用的分区为 /boot, /, /home, /tmp, /var 等分区， 数据磁盘全部容量规划在同一个分区而已。
    - 安装最新的 CentOS 7.x 版

在上面的案例中，案例一是属于小规模的主机系统，因此只要使用预计被淘汰的配备即可进行主机的架设！ 唯一可能需要购买的大概是网卡吧！呵呵！而在案例二中，由于我需要大量的数值运算， 且运算结果的数据非常的庞大，因此就需要比较大的磁盘容量与较佳的网络系统了。 以上的数据请先记得，因为下一章节在实际安装Linux之前，你得先进行主机的规划呀！

2.4 重点回顾

- 新添购计算机硬件配备时，需要考虑的角度有“游戏机/工作机”、“性能/价格比”、“性能/消耗瓦数”、“支持度”等；
- 旧的硬件配备可能由于保存的问题或者是电子零件老化的问题，导致计算机系统非常容易在运行过程中出现不明的死机情况
- Red Hat的硬件支持：https://hardware.redhat.com/?pagename=hcl
- 在Linux系统中，每个设备都被当成一个文件来对待，每个设备都会有设备文件名。
- 磁盘设备文件名通常分为两种，实际SATA/USB设备文件名为/dev/sd[a-p]，而虚拟机的设备可能为/dev/vd[a-p]
- 磁盘的第一个扇区主要记录了两个重要的信息，分别是：（1）主要开机记录区（Master Boot Record, MBR）：可以安装开机管理程序的地方，有446 Bytes （1）分区表（partition table）：记录整颗硬盘分区的状态，有64 Bytes；
- 磁盘的 MBR 分区方式中，主要与延伸分区最多可以有四个，逻辑分区的设备文件名号码，一定由5号开始；
- 如果磁盘容量大于 2TB 以上时，系统会自动使用 GPT 分区方式来处理磁盘分区。
- GPT 分区已经没有延伸与逻辑分区的概念，你可以想像成所有的分区都是主分区！
- 某些操作系统要使用 GPT 分区时，必须要搭配 UEFI 的新型 BIOS 格式才可安装使用。
- 开机的流程由：BIOS-->MBR-->-->boot loader-->核心文件；
- boot loader的功能主要有：提供菜单、载入核心、转交控制权给其他loader
- boot loader可以安装的地点有两个，分别是 MBR 与 boot sector
- Linux操作系统的文件使用目录树系统，与磁盘的对应需要有“挂载”的动作才行；
- 新手的简单分区，建议只要有/及swap两个分区即可

2.5 本章习题

（要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看） 实作题部分：

- 请分析你的家用计算机，以你的硬件配备来计算可能产生的耗电量，最终再以计算出来的总瓦数乘上你可能开机的时间， 以推估出一年你可能会花费多少钱在你的这部主机上面？硬件里面包括 CPU/硬盘/主板/内存/显卡/屏幕等等都会消耗电力，同时电源供应器也会消耗一部份的电力。 若有实际测量工具时，请使用测量结果来计算。若无测量工具，请上网找出每个元件的最大理论消耗功率来计算。

问答题部分：

- 一部计算机主机是否只要 CPU 够快，整体速度就会提高？不见得！一部计算机系统的速度与整体计算机系统的运行有关，每个元件皆会影响计算机的速度！ 这包括了内存、CPU、AGP与显卡速度，硬盘的速度以及其他相关的输入输出接口等等！ 所以，如果您的系统是升级的，那么还得必须要注意各个旧元件是否可以保留， 或者旧的可以用的元件必须要舍弃！
- Linux 对于硬件的要求需要的考虑为何？是否一定要很高的配备才能安装 Linux ？Linux 对于硬件的要求是因“服务种类、服务范围及主机的角色”而定的。例如一部专门用来运算数值解析的 Linux 运算工作站，需要比较强大的 CPU 与足够的 RAM 来进行工作，至于一般家庭用的仅用来做为 ADSL 宽带分享器的 Linux 主机，则只要 P-III 等级的计算机，甚至 P-II 系列的等级，就可以很顺利的运行 Linux 了。
- 一部好的主机在安装之前，最好先进行规划，哪些是必定需要注意的 Linux 主机规划事项？依据上一题的答案内容，我们知道 Linux 对于硬件的要求是“因地制宜”地！所以，要进行 Linux 的安装之前，一定需要规划 Linux 主机的定位与角色！因此， Linux 的主机是否开放网络服务？这部主机的未来规划中，是否需要进行大量的运算？这 部主机是否需要提供很大的硬盘容量来服务客户端的使用？这部主机预计开放的网络服务内容？ 等等，都是需要经过考虑的，尤其未来的“套件选择安装”上面，更需要依据这些规划来设置。
- 请写下下列配备中，在 Linux 的设备文件名： SATA硬盘： CDROM： 打印机： 软盘机：
    - SATA硬盘：/dev/sd[a-d]
    - CDROM：/dev/cdrom
    - 打印机：/dev/lp[0-2]
    - 软盘机：/dev/fd[0-1]
- 目前在个人计算机上面常见的硬盘与主板的连接接口有哪两个？有内置的 SATA 界面与外接式的 USB 界面

2.6 参考资料与延伸阅读

- [1]GUID / GPT 磁盘分区表与 MBR 的限制 wiki 简介： http://zh.wikipedia.org/wiki/GUID磁盘分区表 http://zh.wikipedia.org/wiki/全局唯一标识符
- [2]与 UEFI 界面有关的介绍： Wiki 介绍：http://zh.wikipedia.org/wiki/统一可延伸固件接口 T客帮介绍：http://www.techbang.com/posts/4361-fully-understand-uefi-bios-theory-and-actual-combat-3-liu-xiudian 黄明华先生的介绍：http://www.netadmin.com.tw/article_content.aspx?sn=1501070001&jump=3

2002/04/08：第一次完成吧？ 2003/02/02：重新编排与加入 FAQ 2005/06/04：将旧的文章移动到 这里 2005/06/12：风格修订之外，新增了 Linux 练习机硬件选择与软件安装的建议 2005/06/15：感谢上奇编辑 Tim 兄来信告知一些可能有争议的部分！包括 AthlonXP 已被 Sempron 取代，已经修订！ 2008/07/29：将旧的FC4文章移动到此处。 2008/08/21：将整份文件作个重新整理，移除计概有谈到的硬件部分，增加partition的数据量。 2009/08/06：重新修订习题与解答，尤其一些计概方面的问题将他挪开！ 2015/04/23：将旧的基于 CentOS5 之前的文章移动到此处。 2015/04/28：加入了阅读许久的 UEFI 界面以及 GPT 的相关说明！更厘清了为啥 /boot 不在 /dev/sda1 的位置上啰！

第三章、安装 CentOS7.x

最近更新日期：20//

Linux distributions越作越成熟，所以在安装方面也越来越简单！虽然安装非常的简单， 但是刚刚前一章所谈到的基础认知还是需要了解的，包括MBR/GPT, partition, boot loader, mount, software的选择等等的数据。 这一章鸟哥的安装定义为“一部练习机”，所以安装的方式都是以最简单的方式来处理的。 另外，鸟哥选择的是CentOS 7.x的版本来安装的啦！在内文中，只要标题内含有（Option） 的，代表是鸟哥额外的说明，你应该看看就好，不需要实作喔！^_^

3.1 本练习机的规划--尤其是分区参数

读完主机规划与磁盘分区章节之后，相信你对于安装 Linux 之前要作的事情已经有基本的概念了。唔！并没有读第二章...千万不要这样跳着读，赶紧回去念一念第二章， 了解一下安装前的各种考虑对你Linux的学习会比较好啦！

如果你已经读完第二章了，那么下面就实际针对第二章的介绍来一一规划我们所要安装的练习机了吧！ 请大家注意唷，我们后续的章节与本章的安装都有相关性，所以，请务必要了解到我们这一章的作法喔！

- Linux主机的角色定位： 本主机架设的主要目的在于练习Linux的相关技术，所以几乎所有的数据都想要安装进来。 因此连较耗系统资源的X Window System也必须要包含进来才行。
- 选择的distribution： 由于我们对于Linux的定位为“服务器”的角色，因此选择号称完全相容于商业版RHEL的社群版本， 就是CentOS 7.x版啰。请回到2.3.1 章去获得下载的信息吧！ ^_^。
- 计算机系统硬件配备： 由于虚拟机越来越流行，因此鸟哥这里使用的是 Linux 原生的 KVM 所搭建出来的虚拟硬件环境。对于 Linux 还不熟的朋友来说， 建议你使用2.4 章提到的 virtualbox 来进行练习吧！ 至于鸟哥使用的方式可以参考文末的延伸阅读，里面有许多的文件可参考[1]！ 鸟哥的虚拟机硬件配备如下：
  - CPU等级类别： 通过 Linux 原生的虚拟机管理员的处理，使用本机的 CPU 类型。本机 CPU 为 Intel i7 2600 这颗三、四年前很流行的 CPU 喔！ 至于芯片组则是 KVM 自行设置的喔！
  - 内存： 通过虚拟化技术提供大约 1.2G 左右的内存
  - 硬盘： 使用一颗 40GB 的 VirtI/O 芯片组的磁盘，因此磁盘文件名应该会是 /dev/vda 才对。同时提供一颗 2GB 左右的 IDE 界面的磁盘， 这颗磁盘仅是作为测试之用，并不安装系统！因此还有一颗 /dev/sda 才对喔！
  - 网卡： 使用 bridge （桥接） 的方式设置了对外网卡，网卡同样使用 VirtI/O 的芯片，还好 CentOS 本身就有提供驱动程序， 所以可以直接抓到网卡喔！
  - 显卡（VGA）： 使用的是在 Linux 环境下运行还算顺畅的 QXL 显卡，给予 60M 左右的显示内存。
  - 其他输入/输出设备： 还有仿真光驱、USB鼠标、USB键盘以及 17 英寸屏幕输出等设备喔！
- 磁盘分区的配置 在第二章里面有谈到 MBR 与 GPT 磁盘分区表配置的问题，在目前的 Linux 环境下， 如果你的磁盘没有超过 2TB 的话，那么 Linux 默认是会以 MBR 模式来处理你的分区表的。由于我们仅切出 40GB 的磁盘来玩， 所以默认上会以 MBR 来配置！这鸟哥不喜欢！因为就无法练习新的环境了～因此，我们得在安装的时候加上某些参数， 强迫系统使用 GPT 的分区表来配置我们的磁盘喔！而预计实际分区的情况如下：

| 所需目录/设备 | 磁盘容量 | 文件系统 | 分区格式 | | --- | --- | --- | --- | | BIOS boot | 2MB | 系统自订 | 主分区 | | /boot | 1GB | xfs | 主分区 | | / | 10GB | xfs | LVM 方式 | | /home | 5GB | xfs | LVM 方式 | | swap | 1GB | swap | LVM 方式 |

由于使用 GPT 的关系，因此根本无须考虑主/延伸/逻辑分区的差异。不过，由于 CentOS 默认还是会使用 LVM 的方式来管理你的文件系统， 而且我们后续的章节也会介绍如何管理这东西，因此，我们这次就使用 LVM 管理机制来安装系统看看！

- 开机管理程序（boot loader）： 练习机的开机管理程序使用CentOS 7.x默认的grub2软件，并且安装到MBR上面。 也必须要安装到MBR上面才行！因为我们的硬盘是全部用在Linux上面的啊！ ^_^
- 选择软件： 我们预计这部练习机是要作为服务器用的，同时可能会用到图形接口来管理系统，因此使用的是“含有 X 接口的服务器软件”的软件方式来安装喔！ 要注意的是，从 7.x 开始，默认选择的软件模式会是最小安装！所以千万记得软件安装时，要特别挑选一下才行！
- 检查表单： 最后，你可以使用下面的表格来检查一下，你要安装的数据与实际的硬件是否吻合喔：

```
| 是与否，或详细信息 | 细部项目 |
| --- | --- |
| 是, DVD版 | 01\. 是否已下载且烧录所需的Linux distribution？（DVD或CD） |
| CentOS 7.1, x64 | 02\. Linux distribution的版本为何？（如CentOS 7.1 x86_64版本） |
| x64 | 03\. 硬件等级为何（如i386, x86_64, SPARC等等，以及DVD/CD-ROM） |
| 是, 均为x86_64 | 04\. 前三项安装媒体/操作系统/硬件需求，是否吻合？ |
| 是 | 05\. 硬盘数据是否可以全部被删除？ |
| 已确认分区方式 | 06\. Partition是否做好确认（包括/与swap等容量） | 硬盘数量: 1颗40GB硬盘，
| 有，使用 VirtI/O | 07\. 是否具有特殊的硬件设备（如SCSI磁盘阵列卡等） |
| CentOS 已内置 | 08\. 若有上述特殊硬件，是否已下载驱动程序？ |
| grub2, MBR | 09\. 开机管理程序与安装的位置为何？ |
| 未取得IP参数 | 10\. 网络信息（IP参数等等）是否已取得？ | 未取得IP的情况下，可以套用如下的IP参数
| Server with X | 11\. 所需要的软件有哪些？ |
```

如果上面表单确认过都没有问题的话，那么我们就可以开始来安装咱们的CentOS 7.x x86*64*版本啰！ ^^

3.2 开始安装CentOS 7

由于本章的内容主要是针对安装一部Linux练习机来设置的，所以安装的分区等过程较为简单。 如果你已经不是第一次接触Linux，并且想要架设一部要上线的Linux主机，请务必前往第二章看一下整体规划的想法喔！ 在本章中，你只要依照前一小节的检查表单检查你所需要的安装媒体/硬件/软件信息等等， 然后就能够安装啦！

安装的步骤在各主要Linux distributions都差不多，主要的内容大概是：

1. 调整开机媒体（BIOS）：务必要使用CD或DVD光盘开机，通常需要调整BIOS；
2. 选择安装模式与开机：包括图形接口/命令行等，也可加入特殊参数来开机进入安装画面；
3. 选择语系数据：由于不同地区的键盘按键不同，此时需要调整语系/键盘/鼠标等配备；
4. 软件选择：需要什么样的软件？全部安装还是默认安装即可？
5. 磁盘分区：最重要的项目之一了！记得将刚刚的规划单拿出来设置；
6. 开机管理程序、网络、时区设置与root密码：一些需要的系统基础设置！
7. 安装后的首次设置：安装完毕后还有一些事项要处理，包括使用者、SELinux与防火墙等！

大概就是这样子吧！好了，下面我们就真的要来安装啰！

3.2.1 调整开机媒体（BIOS）与虚拟机创建流程

因为鸟哥是使用虚拟机来做这次的练习，因此是在虚拟机管理员的环境下选择“ Boot Options ”来调整开机顺序！ 基本上，就是类似 BIOS 调整让 CD 作为优先开机设备的意思。至于实体机器的处理方面， 请参考您主板说明书，理论上都有介绍如何调整的问题。

另外，因为 DVD 实在太慢了，所以，比较聪明的朋友或许会将前一章下载的镜像文件通过类似 dd 或者是其他烧录软件， 直接烧录到 U盘上面，然后在 BIOS 里面调整成为可携式设备优先开机的模式，这样就可以使用速度较快的 USB 开机来安装 Linux 了！ windows 系统上面或许可以使用类似 UNetbootin 或者是 ISOtoUSB 等软件来处理。如果你已经有 Linux 的经验与系统，那么可以使用底的方式来处理：

```
# 假设你的 USB 设备为 /dev/sdc ，而 ISO 文件名为 centos7.iso 的话：
[root@study ~]# dd if=centos7.iso of=/dev/sdc
```

上面的过程会跑好长一段时间，时间的长短与你的 USB 速度有关！一般 USB2.0 的写入速度大约不到 10MB 左右，而 USB3.0 可能可以到 50MB 左右～ 因此会等待好几分钟的时间啦！ 写完之后，这颗 USB 就能够拿来作为开机与安装 Linux 之用了！



Tips 一般的主板环境中，使用 USB 2.0 的U盘设备并没有什么问题，他就是被判定为可携式设备。不过如果是 USB3.0 的设备，那主板可能会将该设备判断成为一颗磁盘！ 所以在 BIOS 的设置中，你可能得要使用磁盘开机，并将这颗 USB “磁盘”指定为第一优先开机，这样才能够使用这颗 U盘来安装 Linux 喔！

如果你暂时找不到主板说明书，那也没关系！当你的计算机重新开机后，看到屏幕上面会有几个文字告诉你如何进入设置 （Setting） 模式中！ 一般常用的进入按钮大概都是“ Del ”按键，或者是“ F2 ”功能键，按下之后就可以看到 BIOS 的画面了！ 大概选择关键字为“ Boot ”的项目，就能够找到开机顺序的项目啰！

在调整完BIOS内的开机设备的顺序后，理论上你的主机已经可使用可开机光盘来开机了！ 如果发生一些错误讯息导致无法以CentOS 7.x DVD来开机，很可能是由于：1）计算机硬件不支持； 2）光驱会挑片； 3）光盘片有问题； 如果是这样，那么建议你再仔细的确认一下你的硬件是否有超频？或者其他不正常的现象。 另外，你的光盘来源也需要再次的确认！

- 在 Linux KVM 上面创建虚拟机的流程

如果你已经在实体机器上面创建好 CentOS 7 了，然后想要依照我们这个基础篇的内容来实验一下学习的进度，那么可以使用下面的流程来创建与课程相仿的硬盘喔！ 创建流程不会很困难，瞧一瞧即可！

首先，你得从“应用程序”里面的“系统工具”找到“虚拟机管理员”，点下他就会出现如下的图示：

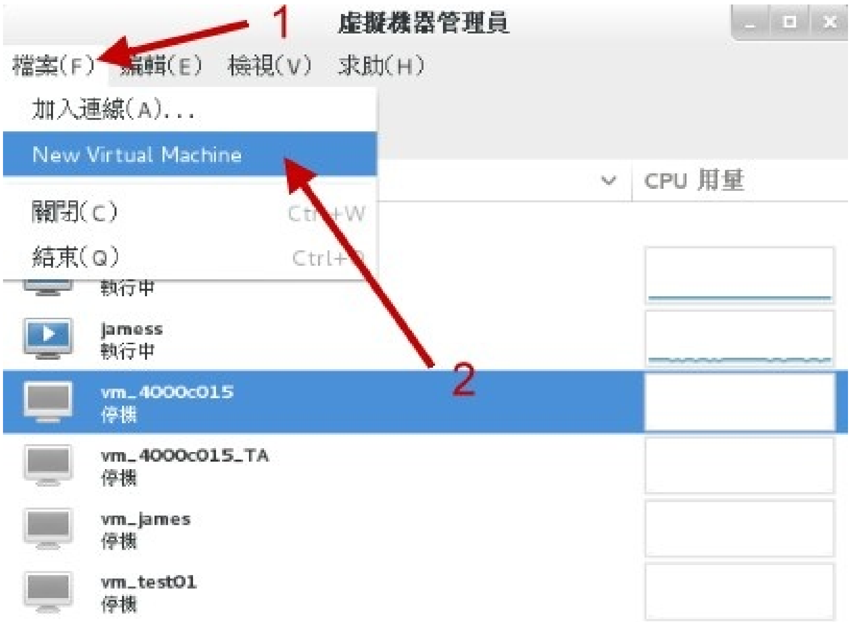


图3.2.1、启动虚拟机管理员示意图

因为我们是想要创建新的虚拟机，因此你要像上图那样，点选“文件”然后点选“ New Virtual Machine ”，接下来就能够看到如下图的模样来创建新机器！



图3.2.2、选择使用光盘来安装，并实际选择 CentOS 镜像文件所在

如上图所示，左图可以让你选择这个新的机器安装的时候，要安装的是哪个来源媒体，包括直接从网络来源安装、从硬盘安装等等。 我们当然是选择光盘镜像文件啰！按下一步就会进入选择光盘镜像文件的文件名～这时请按“浏览”并且选择“文件系统”，再慢慢一个一个选择即可！ 之后就继续下一步吧！



图3.2.3、设置内存容量、CPU数量、磁盘容量等重要机器设置

接下来如上图所示，你可以挑选内存容量、CPU 颗数以及磁盘的容量等等。比较有趣的地方是，你会看到上图右侧鸟哥写了 40G 的容量， 但可用容量只有 28G 耶～这样有没有关系？当然没关系！现在的虚拟机的磁盘机制，大多使用 qcow2 这个虚拟磁盘格式，这种格式是“用多少纪录多少”喔， 与你的实际使用量有关。既然我们才刚刚要使用，所以这个虚拟磁盘当然没有数据，既然没有数据需要写入，那就不会占用到实际的磁盘容量了！ 尽量用！没关系！^_^



图3.2.4、使用桥接的功能设置网络

在出现的画面中，选择“进阶选项”之后，挑选主机设备设置，然后点选桥接功能，如此一来才有办法让你的虚拟机网卡具有直接对外的功能喔！ 同时如果你想要改设置的话，那么可以勾选“在安装前自动组态”的圈圈，之后按完成会出现如下图所示：

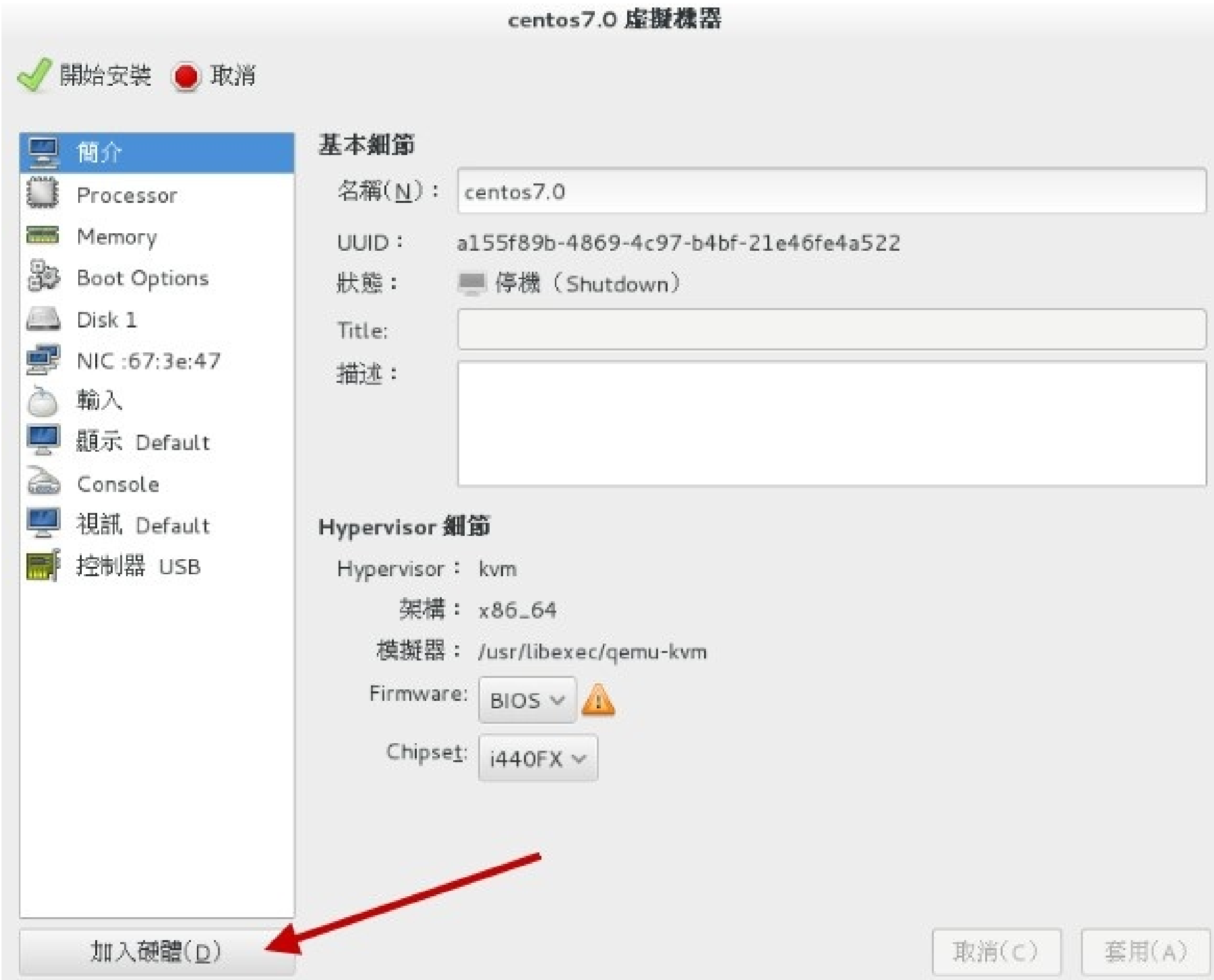


图
3.2.5、设置完成的示意图

从上图 3.2.5 当中，我们可以看到这部机器的相关硬件配备喔！不过，竟然没有发现光驱耶！真怪！那请按下上图中指标指的地方， 加入一个新硬件！新硬件增加的示意图如下所示：

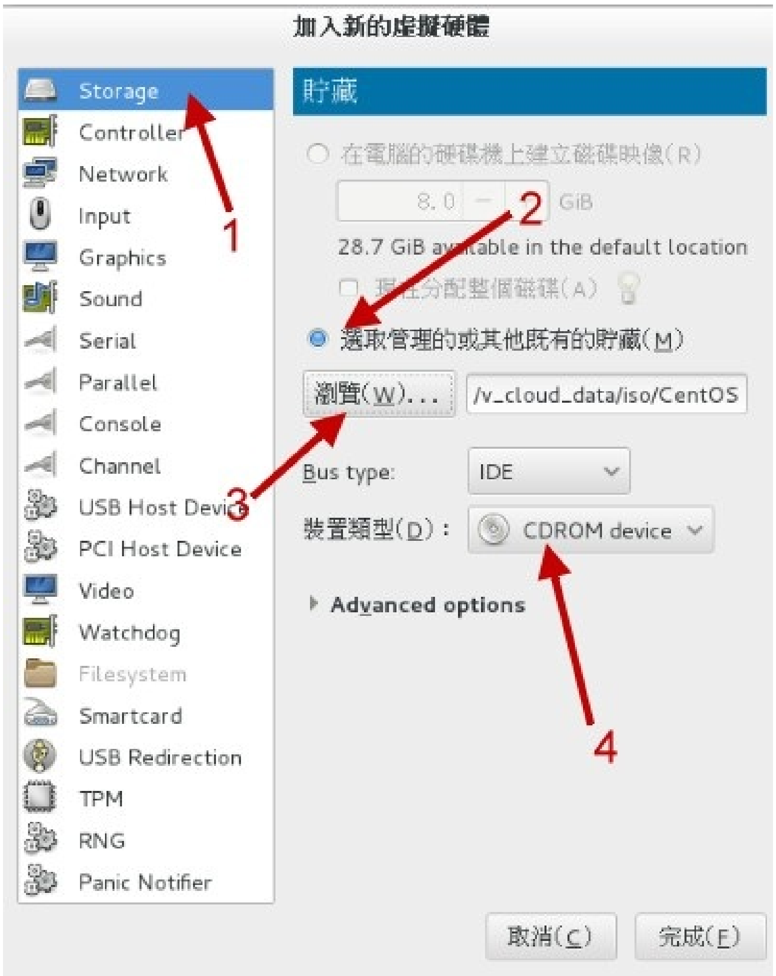


图3.2.6、新增硬件示意图

如上图所示，我们来创建一个 IDE 接口的光盘，并且将光盘镜像文件加入其中！加入完成之后按下“完成”即可出现如下的最终画面了！

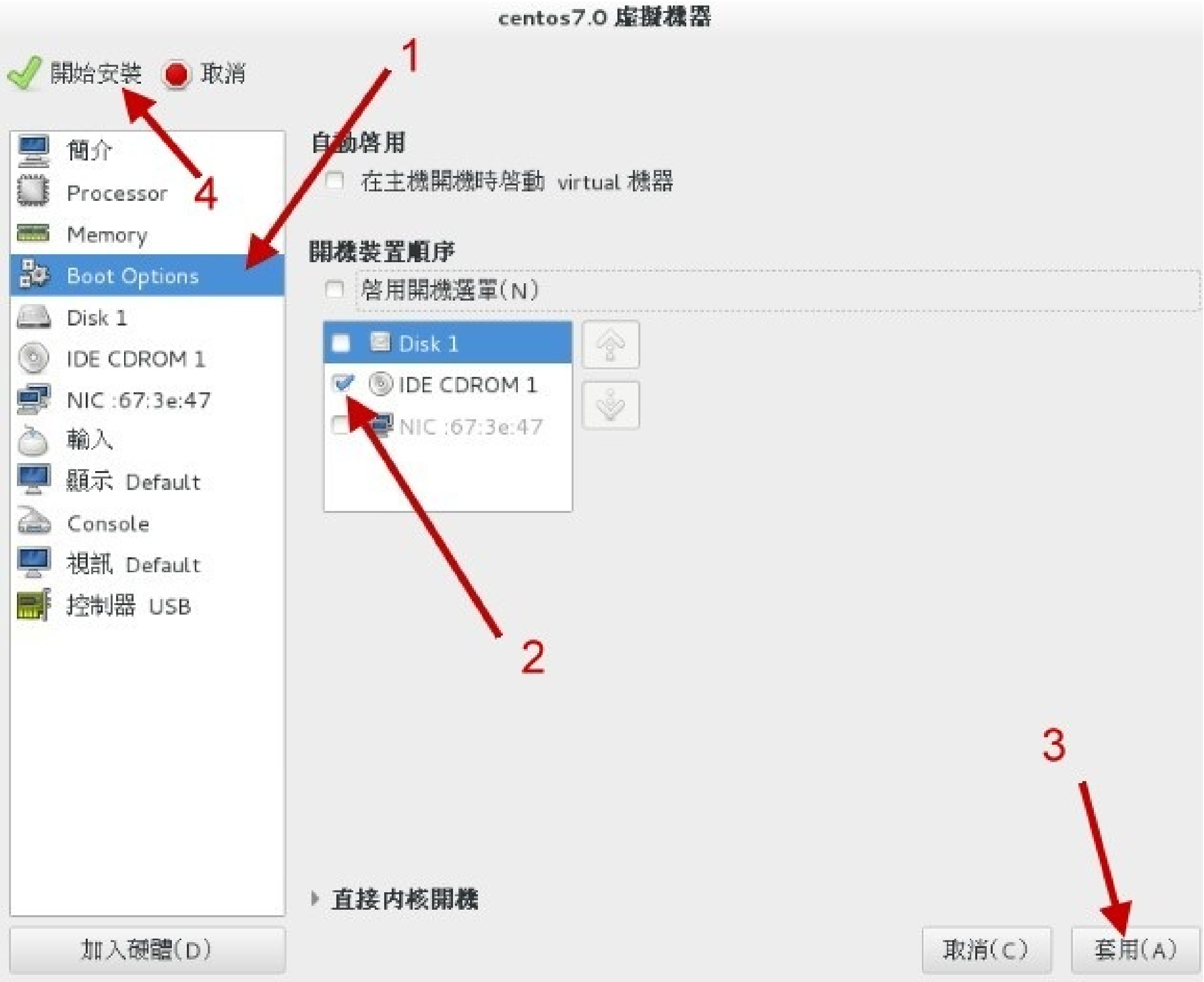


图 3.2.7、虚拟机最终创建完成示意图

这时你的虚拟机已经跟鸟哥的差不多了！按下“开始安装”就能够取得与鸟哥在下列提供的各样设置啰！



Tips 为了方便维护与管理，鸟哥的虚拟机实际上是使用 Gocloud （http://www.gocloud.com.tw/） 虚拟计算机教室系统所创建的！因此上述的流程与鸟哥实际创建的虚拟机， 会有一些些的差异～不过差异不大就是了！这里要先跟大家解释一下！

3.2.2 选择安装模式与开机 - inst.gpt

如果一切都顺利没问题的话，那么使用光盘镜像文件开机后，就会出现如下画面：

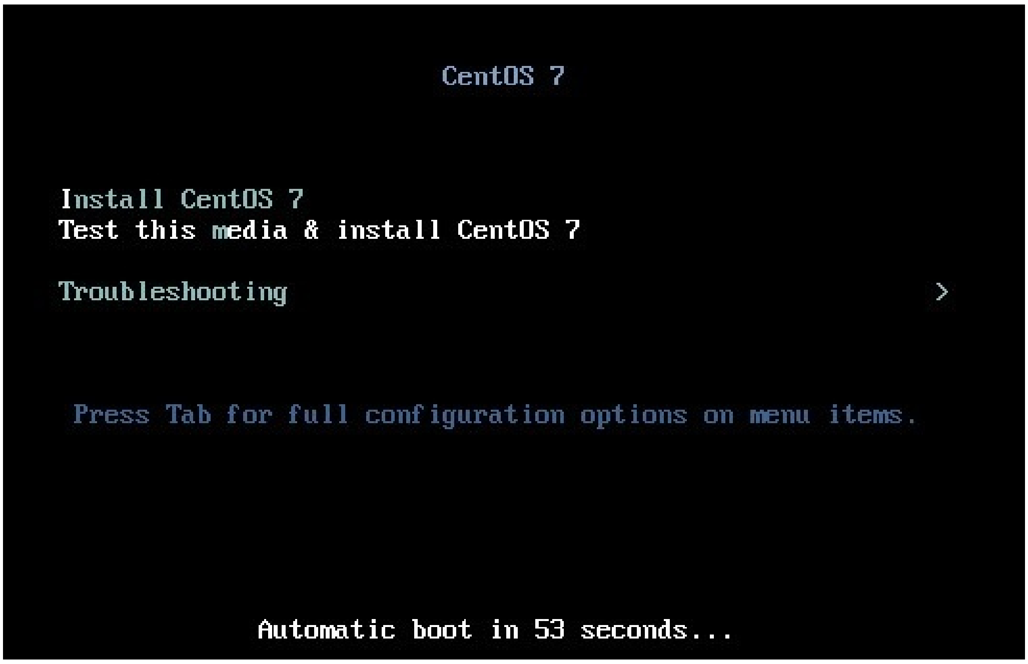


图3.2.8、光盘开机后安装画面之选择

你有 60 秒的时间可以选择不同的操作模式，从上而下分别是：

1. 正常安装 CentOS 7 的流程；
2. 测试此光盘后再进入 CentOS 7 的流程；
3. 进入除错模式！选择此模式会出现更多的选项，分别是：
    - 以基本图形接口安装 CentOS 7 （使用标准显卡来设置安装流程图示）；
    - 救援 CentOS 系统
    - 执行内存测试 （Run a memory test）
    - 由本机磁盘正常开机，不由光盘开机

基本上，除非你的硬件系统有问题，包括拥有比较特别的图形显卡等等，否则使用正常的 CentOS 7 流程即可！ 那如果你怀疑这片光盘有问题，就可以选择测试光盘后再进入 CentOS 7 安装的程序。如果你确信此光盘没问题，就不要测试了！ 不过如果你不在乎花费一、两分钟的时间去测试看看光盘片有没有问题，就使用测试后安装的流程啊！不过要进入安装程序前先等等， 先进行下面的流程再继续。

- 加入强制使用 GPT 分区表的安装参数

如前所述，如果磁盘容量小于 2TB 的话，系统默认会使用 MBR 模式来安装！鸟哥的虚拟机仅有 40GB 的磁盘容量，所以默认肯定会用 MBR 模式来安装的啊！那如果想要强制使用 GPT 分区表的话，你就得要这样作：

1. 使用方向键，将图 3.2.8 的光标移动到“ Install CentOS 7 ”的项目中
2. 按下键盘的 [Tab] 按钮，让光标跑到画面最下方等待输入额外的核心参数
3. 在出现的画面中，输入如下画面的数据 （注意，各个项目要有空格，最后一个是光标本身而非底线）

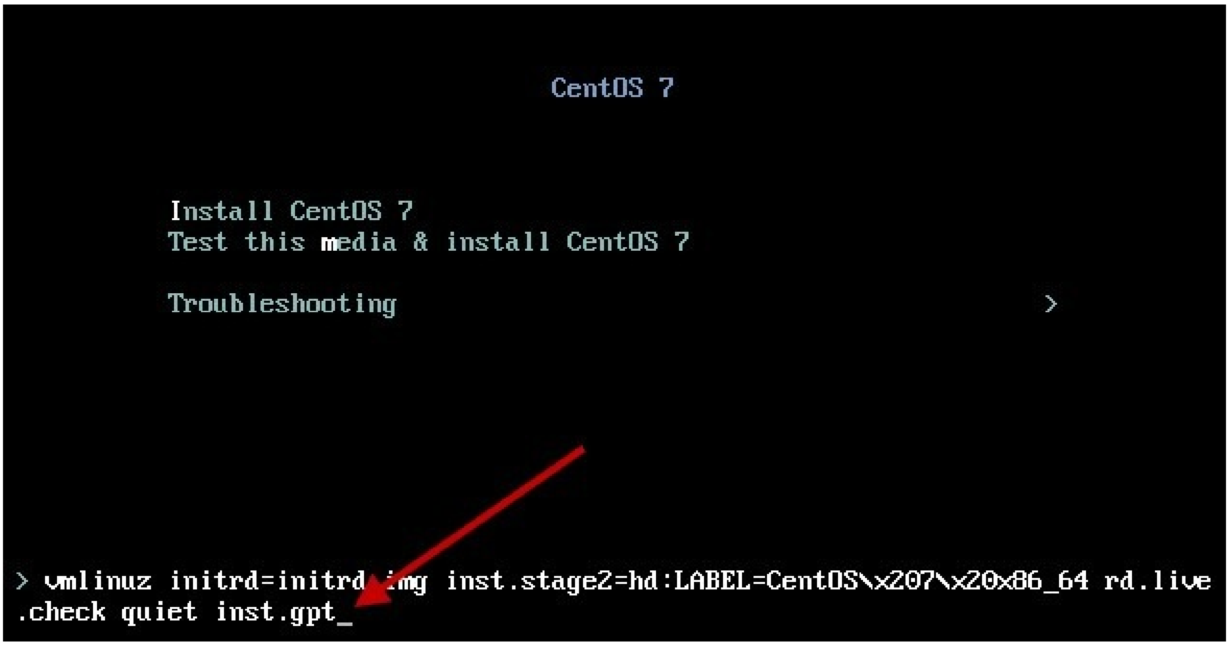


图3.2.9、加入额外的核心参数修改安装程序

其实重点就是输入“ inst.gpt ”这个关键字！输入之后系统会跑过一段侦测的画面， 这段侦测的流程依据你的光驱速度、硬件复杂度而有不同。反正，就是等待个几秒钟到一、两分钟就是了！画面如下所示：

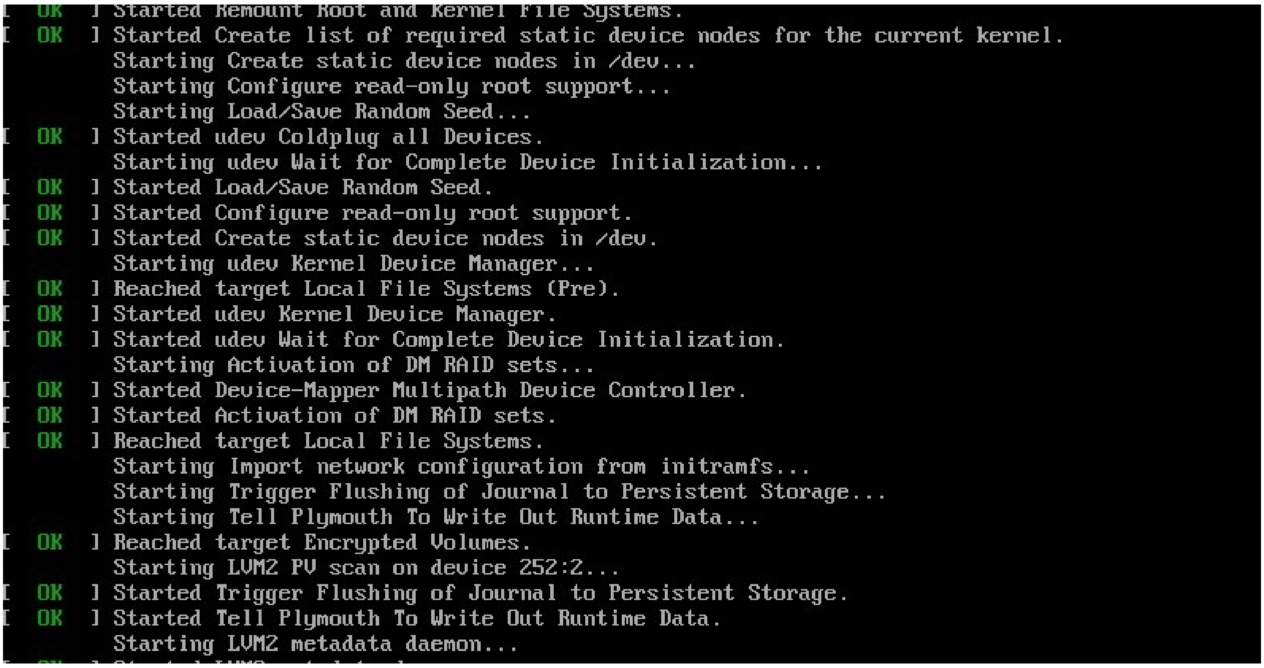


图3.2.10、安装程序的侦测系统过程

进入安装流程的第一个画面就是选择你熟悉的语系啰！这个选择还挺重要的！因为未来默认的语系、默认用户选择的环境等， 都跟这里有关～当然未来是可以改变的～如下图所示，你可以依据箭头的指示选择我们台湾惯用的繁体中文字！然后就可以按下“继续”来处理喔！

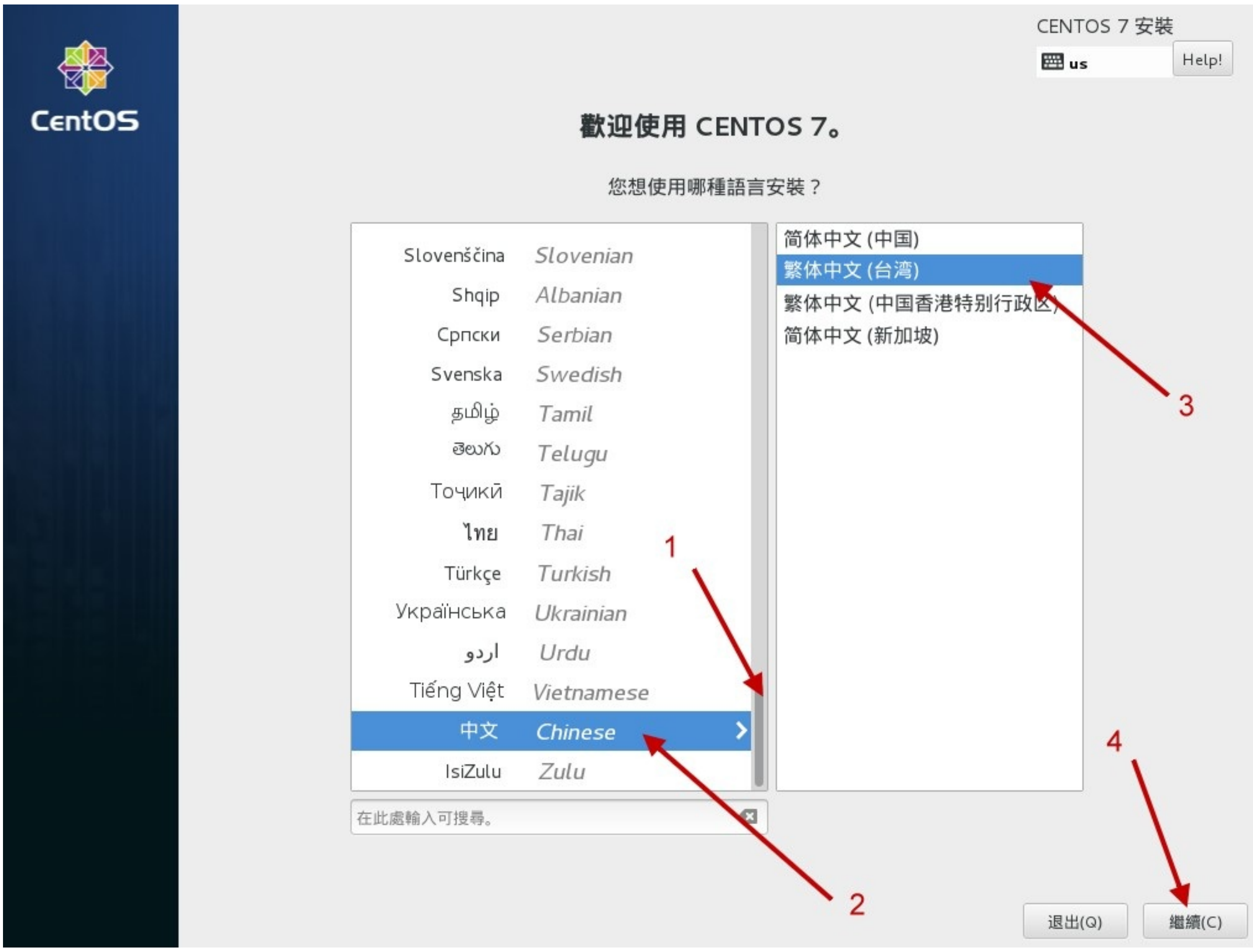


图3.2.11、选择安装程序的语系显示

在 CentOS 7 的安装流程中，已经将所有的挑选流程以按钮形式通通集中在第一页了！如下图所示，所以你可以在同一个画面中看完所有的设置， 也可以跳着修改各个设置，不用被制约一项一项处理喔！下面我们就来谈谈每一个项目的设置方式吧！



图3.2.12、统一按钮展示的安装画面

3.2.3 在地设置之时区、语系与键盘配置

按下 图 3.2.12 画面当中的“在地设置”项目内的“日期时间”后，会出现如下的画面：



图3.2.13、时区挑选的项目示意图

你可以直接在世界地图上面选择到你想要的时区位置，也可以在画面中“区域、城市”的下拉式菜单选择你的城市即可。 如果日期与时间不对，可以在画面中箭头指的 2, 3 处分别修改。虽然有网络的时间自订修订功能，不过因为我们的网络尚未设置好， 所以画面中的箭头 5 无法顺利打开就是了。处理完毕后，按下左上方箭头 4 指的“完成”按钮，即可回到 图 3.2.12 中。



Tips 说实在的，我们这些老人家以前接触的画面，确认钮通常在右下方。第一次接触 CentOS 7 的安装画面时，花了将近一分钟去找确认按钮耶！ 还以为程序出错了！后来才发现在左上方～这...真是欺负老人的设计吗？哈哈哈哈！

时区选择之后，接下来请点选 图 3.2.12 内的“键盘配置”，出现的画面如下：

图3.2.14、键盘配置项目

这个很重要喔！因为我们需要输入中文，所以常常打字会在中/英文之间切换。过去我们经常使用的键盘配置是“ Ctrl + 空白 ”按钮，或者是“ Ctrl + Shift ”按钮， 不过这一版的窗口接口，默认并没有提供任何的切换按钮～所以这里得要预先来设置一下比较妥当。如图中的箭头顺

序去调整， 不过鸟哥一直找不到习惯的“ ctrl + 空白 ”的组合，只好用次习惯的“ Ctrl + Shift ”组合了！确认后可以按完成按钮即可。 不过，如果你想要有其他的输入语系的话，可以选择画面中左下方用圈圈勾起来的地方，按下去就会出现如下画面：



图3.2.15、新增其他语系的键盘配置

竟然还有三种特殊的台湾语系键盘配置规格耶！好有趣！有需要的朋友可以选择看看！至于“语系支持”的画面则与 图 3.2.11 相同， 所以这里就不多说了！

3.2.4 安装来源设置与软件选择

回到 图 3.2.12 后，按下“安装来源”按钮之后，你会得到如下的画面：

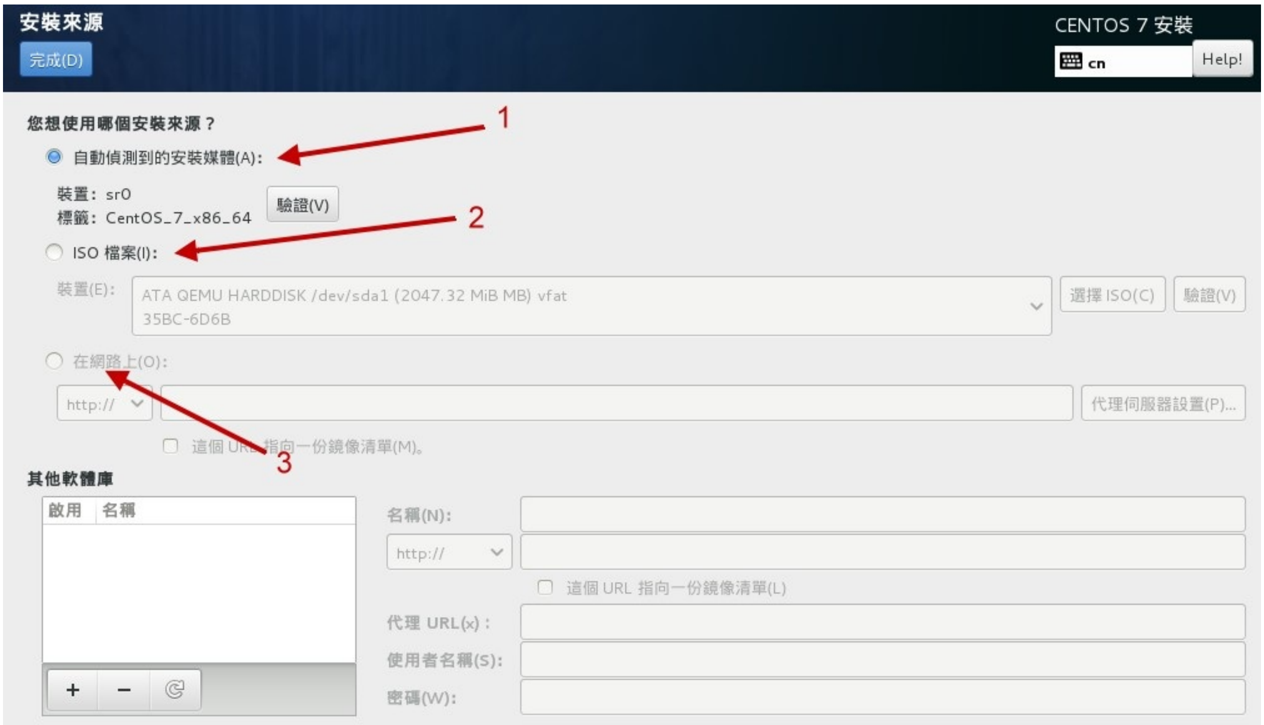


图3.2.16、挑选准备要被安装的软件所在的媒体

因为我们是使用光盘开机，同时还没有设置网络，因此默认就会选择光盘片 （sr0 所在的设备）。如果你的主机系统当中还有其他安装程序认识的磁盘文件系统， 那么由于该磁盘也可能会放置镜像文件啊，所以该镜像文件也能够提供软件的安装，因此就有如同上图的“ISO文件”的选择项目。最后， 如果你的安装程序已经预先设置好网络了，那么就可以选择“在网络上”的项目，并且填写正确的网址 （URL） ，那么安装程序就可以直接从网络上面下载安装了！



Tips 其实如果区域网络里面你可以自己设置一个安装服务器的话，那么使用网络安装的速度恐怕会比其他方式快速喔！毕竟 giga 网络速度可达到 100MBytes/s 的读写， 这个速度 DVD 或 USB 2.0 都远远不及啊！^_^

按下完成并回到图 3.2.12 之后，就得要选择“软件选择”的画面了！如下所示：

图3.2.17、选择安装的软件数据为哪些

因为默认是“最小型安装”的模式，这种模式只安装最简单的功能，很适合高手慢慢搭建自己的环境之用。 但是我们是初学者啊～没有图形接口来看看实在有点怪！所以建议可以选择如下的项目：

- 含有 GUI 的服务器 （GUI就是使用者图形接口啰！默认搭载 GNOME）
- GNOME 桌面环境： Linux 常见的图形接口

- KDE Plasma Workspaces：另一套常见的图形接口

上面这几个设置拥有图形接口，鸟哥这里主要是以“GUI 服务器”作为介绍喔！选择完毕之后按下完成， 安装程序会开始检查光盘里面有没有你所挑选的软件存在，而且解决软件相依性的检查 （就是将你所选择的大项目下面的其他支持软件通通载入） ，之后就会再次的回到 图 3.2.12 的画面中。

3.2.5 磁盘分区与文件系统设置

再来就是我们的重头戏，当然就是磁盘分区啦！由 图 3.2.12 当中，点选“系统”项目下的“安装目的地”区块，点选之后会进入如下画面中：



图3.2.18、选择要安装 Linux 的硬盘，并选择手动分区模式

由于鸟哥的虚拟机系统共有两颗硬盘，因此安装的时候你得要特别选择正确的硬盘才能够顺利的安装喔！所以如上图 1 号箭头所指， 点选之后就会出现打勾的符号啰！因为我们要学习分区的方式，不要让系统自动分区，因此请点选 2 号箭头所指处：“我将配置分页”的项目。点选完毕后按下“完成”，即可出现如下的磁盘分区画面喔！

图3.2.19、删除已经存在系统当中的分区

其实鸟哥故意将硬盘先乱安装一套系统，然后再安装 CentOS7 的，就是为了要在这里展示给各位朋友们瞧一瞧，如何在安装时观察与删除分区啊！ 如上图所示，你会发现到 1 号箭头处有个操作系统名称，点选该名称 （你的系统可能不会有这个项目，也有可能是其他项目！不过，如果是全新硬盘， 你就可以略过这个部份了） ，他就会出现该系统拥有的分区。依序分别点选下面的 /boot, /, swap 三个项目，然后点选 3 号箭头处的减号“ - ”， 就可以删除掉该分区了！删除的时候会出现如下的警告窗口喔！



图3.2.20、删除分区时出现的警告窗口示意图

因为前一个系统鸟哥安装的也是旧版的 CentOS 6.x 的版本，所以 CentOS7 可以自动抓到所有该系统的挂载点～于是就会出现如上所示的图示， 会特别询问你要不要同时删出其他的分区。我们原本有 3 个分区需要删除，点选上图 1 号箭头然后按下“删除它”，嘿嘿！三个分区全部会被删除干净！ 之后就会回 图 3.2.19 的画面中了！之后你就可以开始创建文件系统啰！同时请注意，分区的时候请参考本章 3.1 小节的介绍， 根据该小节的建议去设置好分区喔！下面我们先来制作第一个 GPT 分区表最好要拥有的 BIOS boot 分区，如下所示：

图3.2.21、创建 BIOS boot 分区的示意图

先点选 1 号箭头处的菜单，不要使用默认的 LVM 喔！请点选“标准分区区”的项目，并按下 2 号箭头的“ + ”符号，就会出现中间的弹出式窗口， 在该窗口中 3 号箭头处，点选下拉式菜单然后选择你在画面中看到的 biosboot 项目 （不要手动输入画面中的文字，请使用既有的菜单来挑选喔！），同时输入大约 2M 的容量，按下“新增挂载点”后， 就会整理出该分区的详细数据，如下图所示：

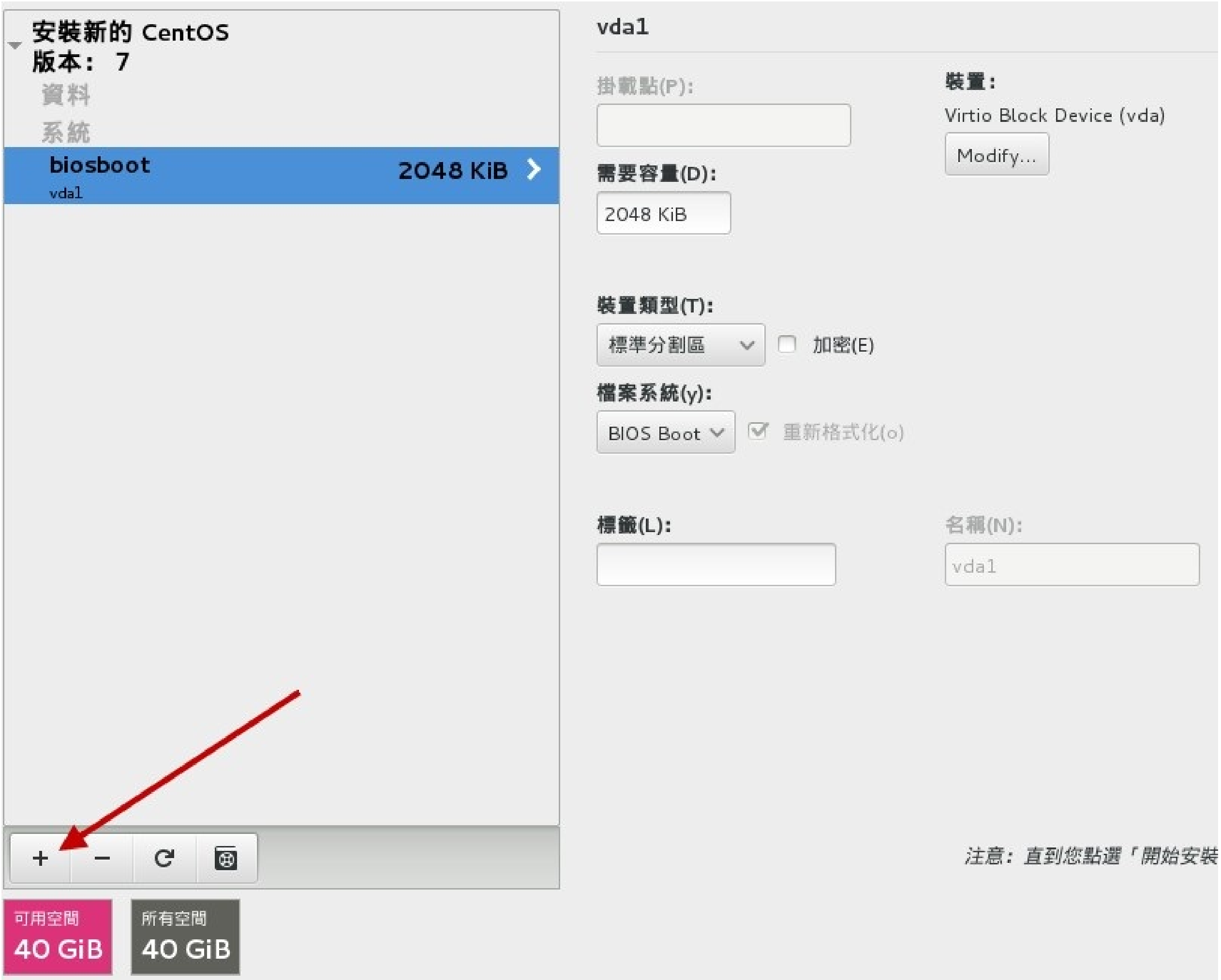


图3.2.22、单一分区分区完成详细项目示意图

如上图所示，画面的右边就是 biosboot 分区的详细部份！由于是 bios 使用，因此没有挂载点（你看画面中该字段是空空如也的！）。 同时文件系统的字段部份也是会变成“BIOS Boot”的关键字！并不会是 Linux 的文件系统啦！接下来，我们要来设置其他的分区了！ 所以如上图所示，请按下“ + ”符号吧！下面的示意图鸟哥就不全图撷取，只抓出弹出式窗口的内容来给大家瞧瞧喔！

另外，图中的“设备类型”其实共有 3 种，我们的练习机实际使用标准分区与 LVM 而已。那三种设备类型的意义分别如下：

- 标准分区区：就是我们一直谈的分区啊！类似 /dev/vda1 之类的分区就是了。
- LVM：这是一种可以弹性增加/削减文件系统容量的设备设置，我们会在后面的章节持续介绍 LVM 这个有趣的东西！
- LVM 紧张供应：这个名词翻译的超奇怪的！其实这个是 LVM 的进阶版！与传统 LVM 直接分配固定的容量不同， 这个“ LVM 紧张供应”的项目，可以让你在使用多少容量才分配磁盘多少容量给你，所以如果 LVM 设备内的数据量较少，那么你的磁盘其实还可以作更多的数据储存！ 而不会被平白无故的占用！这部份我们也在后续谈到 LVM 的时候再来强调！

另外，图中的文件系统就是实际“格式化”的时候，我们可以格式化成什么文件系统的意思。下面分别谈谈各个文件系统项目 （详细的项目会在后续章节说明）

- ext2/ext3/ext4：Linux早期适用的文件系统类型。由于ext3/ext4文件系统多了日志的记录， 对于系统的复原比较快速。不过由于磁盘容量越来越大，ext 家族似乎有点挡不住了～所以除非你有特殊的设置需求，否则近来比较少使用 ext4 项目了！
- swap：就是磁盘仿真成为内存，由于swap并不会使用到目录树的挂载，所以用swap就不需要指定挂载点喔。
- BIOS Boot：就是 GPT 分区表可能会使用到的项目，若你使用 MBR 分区，那就不需要这个项目了！
- xfs：这个是目前 CentOS 默认的文件系统，最早是由大型服务器所开发出来的！ 他对于大容量的磁盘管理非常好，而且格式化的时候速度相当快，很适合当今动不动就是好几个 TB 的磁盘的环境喔！因此我们主要用这玩意儿！
- vfat：同时被Linux与Windows所支持的文件系统类型。如果你的主机硬盘内同时存在Windows与Linux操作系统，为了数据的交换， 确实可以创建一个vfat的文件系统喔！

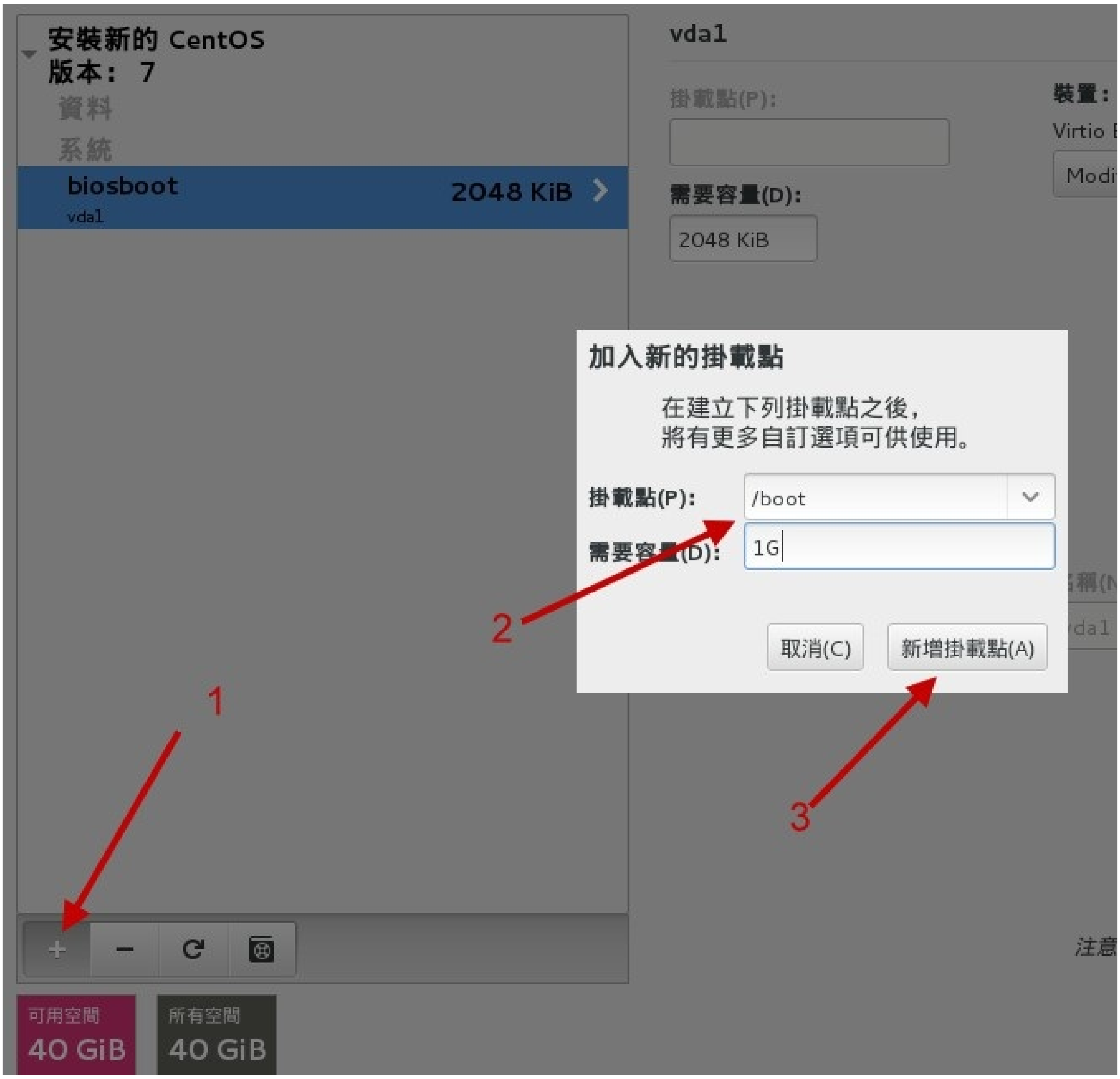


图3.2.23、创建 /boot 分区的示意图

依据 3.1 小节的建议，接下来是创建 /boot 挂载点的文件系统。容量的部份你可以输入 1G 或者是 1024M 都可以！有简单的单位较佳。 然后按下新增吧！就会回到类似 图 3.2.22 的画面喔！接下来依序创建另外所需要的根目录“ / ”的分区吧！

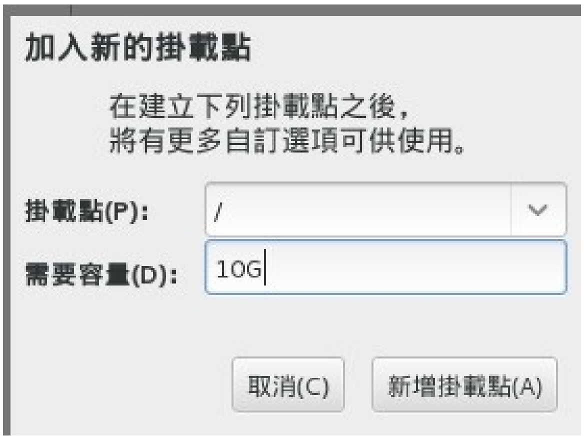


图3.2.24、创建根目录 / 的分区

如上图所示，就输入根目录的容量吧！依据 3.1 小节的建议给予 10G 的容量。接下来要注意喔，我们的 /, /home, swap 都希望使用 CentOS 提供的 LVM 管理方式，因此当你按下上图的“新增挂载点”之后，回到下面的详细设置项目时，得要更改一下相关的项目才行！如下所示：

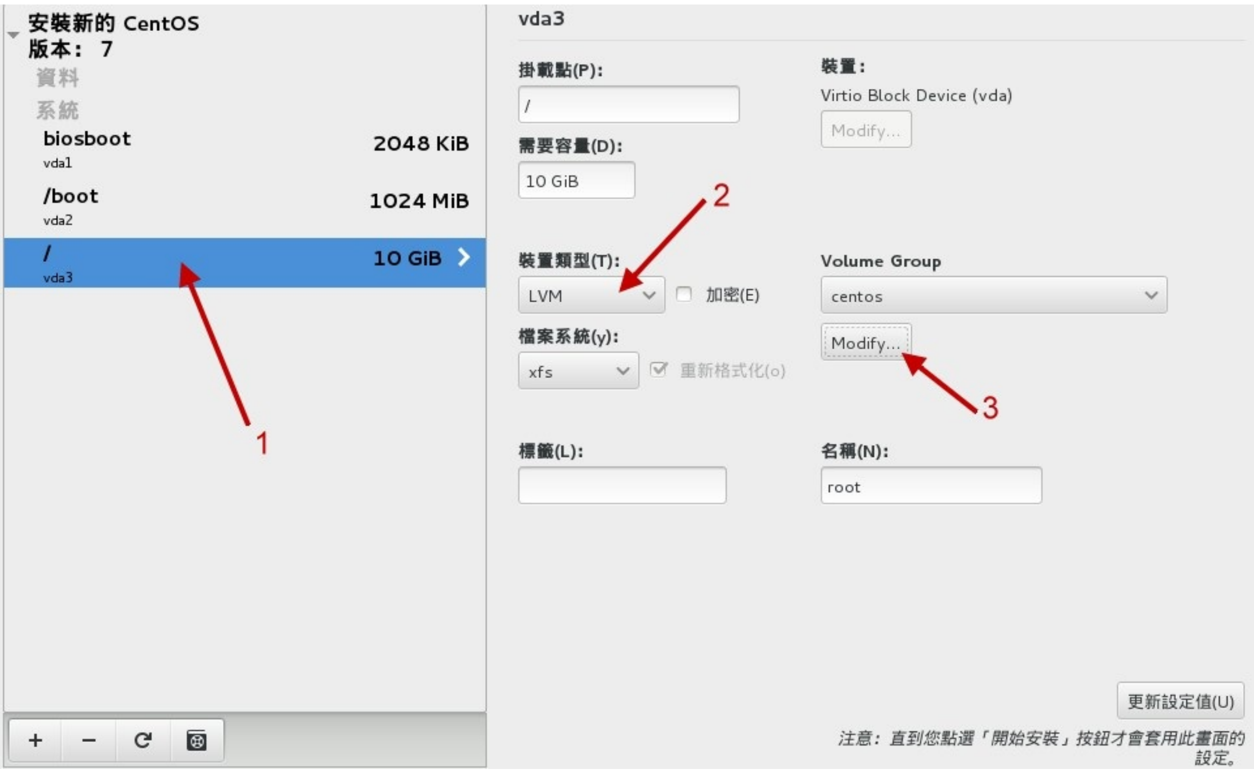


图3.2.25、将设备类型改为 LVM 的管理机制

如上图所示，你得先确认 1 号箭头指的地方为 / 才对，然后点选 2 号箭头处，将他改为“LVM”才好。由于 LVM 默认会取一个名为 centos 的 LVM 设备， 因此该项目不用修改！只要按下 3 号箭头处的“ Modify（更改） ”即可。接下来会出现如下的画面，要让你处理 LVM 的相关设置！



图3.2.26、修改与设置 LVM 设备的容量

再次说明，我们这里是要创建一个让你在未来可以持续练习的练习机环境，因此不建议将分区用完！所以，如上图所示， 1 号箭头处请选择“固定”容量，然后填入“ 30G ”左右的容量，这样我们就还有剩下将近 10G 的容量可以继续未来的章节内容练习。 其他的就保留默认值，点选“储存”来确定吧！然后回到类似 图 3.2.23 的画面，继续点选“ + ”来持续新增分区，如下所示：

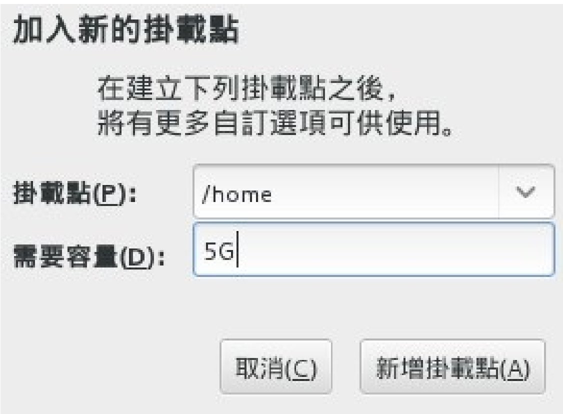


图3.2.27、创建 /home 分区

创建好 /home 分区之后，同样需要调整 LVM 设备才行，因此在你按下上图的“新增挂载点”之后，回到下面的画面来处理处理！

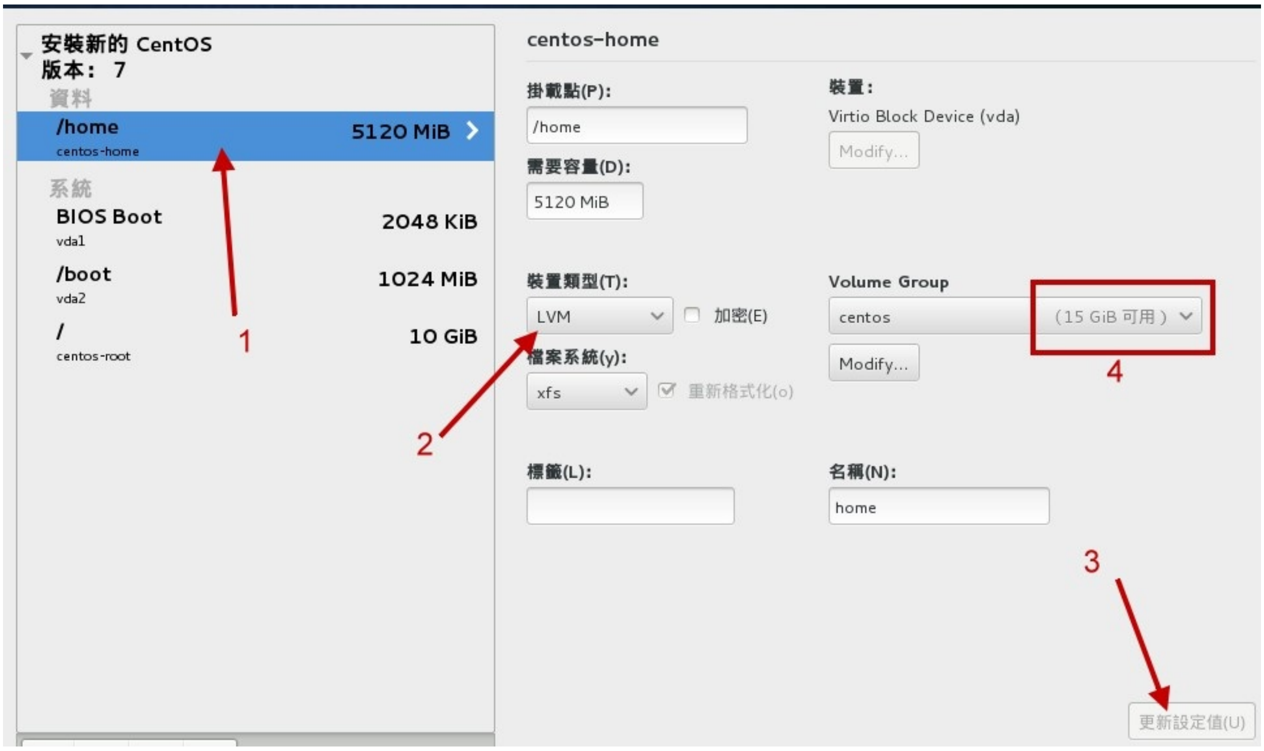


图3.2.28、调整 /home 也使用 LVM 设备

如上图所示，确定 1 号箭头是 /home ，然后选择 2 号箭头成为 LVM，之后确定 4 号箭头还有剩余容量 （也是为了未来要练习之用）， 之后就可以按下 3 号箭头的变更设置来确认啰！其实要先按 3 号箭头，4 号区块才会顺利显示啦！ ^_^

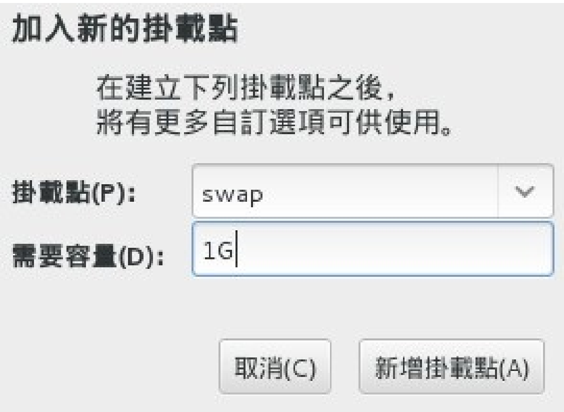


图3.2.29、创建 swap 分区

swap 是当实体内存容量不够用时，可以拿这个部份来存放内存中较少被使用的程序项目。以前都建议 swap 需要内存的 2 倍较佳。 不过现在的内存都够大了，swap 虽然最好还是保持存在比较好，不过也不需要太大啦！大约 1~2GB 就好了。老实说，如果你的系统竟然会使用到 swap， 那代表...钱花的不够多！继续扩充内存啦！



Tips swap内存交换空间的功能是：当有数据被存放在实体内存里面，但是这些数据又不是常被CPU所取用时，那么这些不常被使用的程序将会被丢到硬盘的swap交换空间当中，而将速度较快的实体内存空间释放出来给真正需要的程序使用！所以，如果你的系统不很忙，而内存又很大，自然不需要swap啰。

图3.2.30、调整 swap 也使用 LVM 设备

如上图所示，我们也需要 swap 使用 LVM，请按照箭头依序处理各个项目吧！上述的动作做完之后，我们的分区就准备妥当了！接下来，看看你的分区是否与下图类似！需要有 /home, /boot, /, swap 等项目。

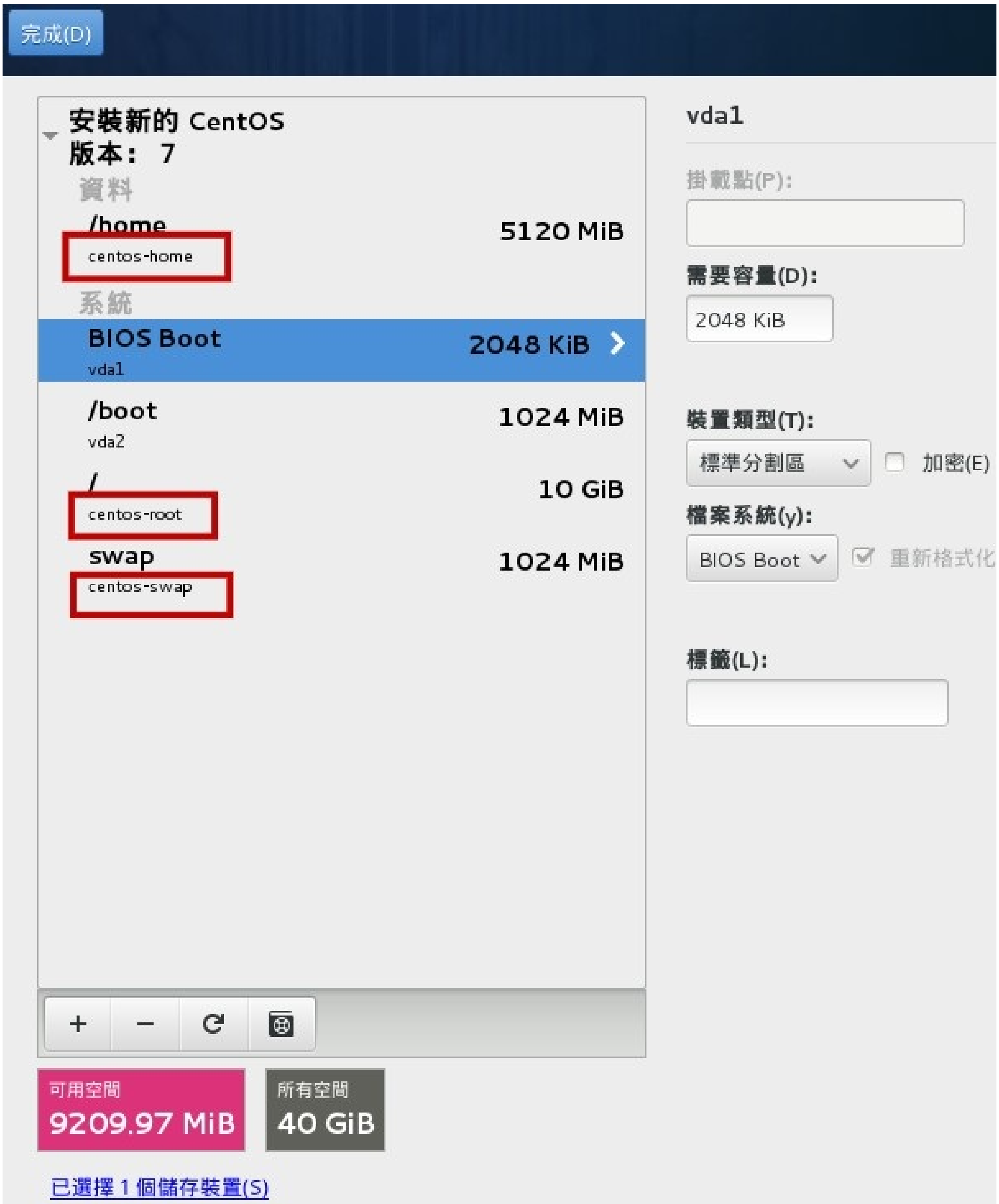


图
3.2.31、完成分区之后的示意图

如上图所示，仔细看一下左下角的两个方块，可用空间的部份还有剩下大约 9GB 左右，这样才对喔！ 如果一切顺利正常，按下上图左上方的“完成”，系统会出现一个警告窗口，提醒你是否要真的进行这样的分区与格式化的动作，如下图所示：



图3.2.32、是否确定分区正确的示意图

上图中你可以特别观察一下分区表的类型，可以发现方框圈起来的地方，删除了 MSDOS 而创建了 GPT ！嘿嘿！没错！是我们要的！所以，按下“接受变更”吧！ 之后就会回到 图 3.2.12 的画面啰！

3.2.6 核心管理与网络设置

回到 图 3.2.12 的画面后，点选“系统”下的“KDUMP”项目，这个项目主要在处理，当 Linux 系统因为核心问题导致的死机事件时， 会将该死机事件的内存内数据储存出来的一项特色！不过，这个特色似乎比较偏向核心开发者在除错之用～如果你有需要的话，也可以启动它！ 若不需要，也能够关闭它，对系统的影响似乎并不太大。所以，如下图所示，点选之后，鸟哥是使用“启用”的默认值，并没有特别取消掉这项目就是了。



图3.2.33、KDUMP 的挑选示意图

再次回到 图 3.2.12 的画面点选“系统”下的“网络&主机名称”的设置，会出现如下图所示画面：

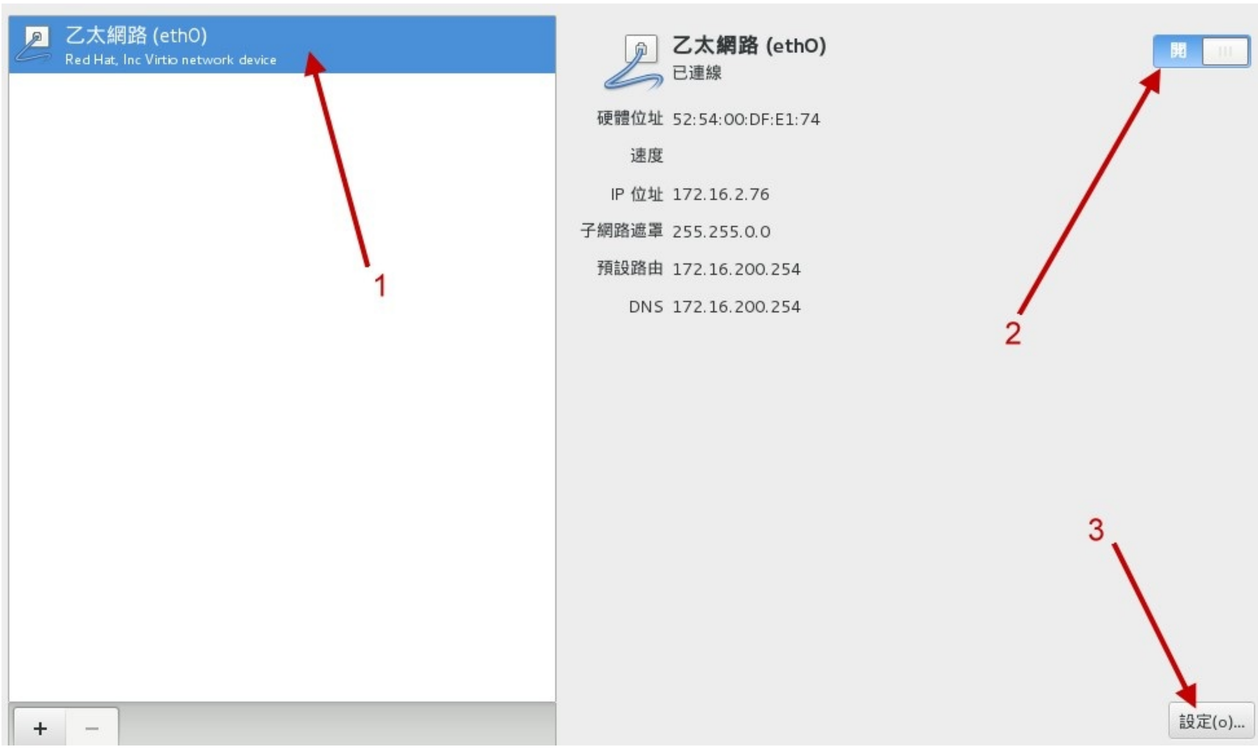


图3.2.34、网络设置示意图

因为鸟哥这边使用的是虚拟机，因此看到的网卡就会是旧式的 eth0 之类的网卡代号。如果是实体网卡，那你可能会看到类似 p1p1, em1 等等比较特殊的网卡代号！ 这是因为新的设计中，它是以网卡安插的插槽来作为网卡名称的由来 [2]，这部份未来我们在网络再来谈！这里先知道一下即可。

上图中先选择正确的网卡，然后在 2 号箭头处选择“开”之后，3 号箭头处才能够开始设置！现在请按下“设置”项目，然后参考 3.1 小节的介绍，来给予一组特别的 IP 吧！

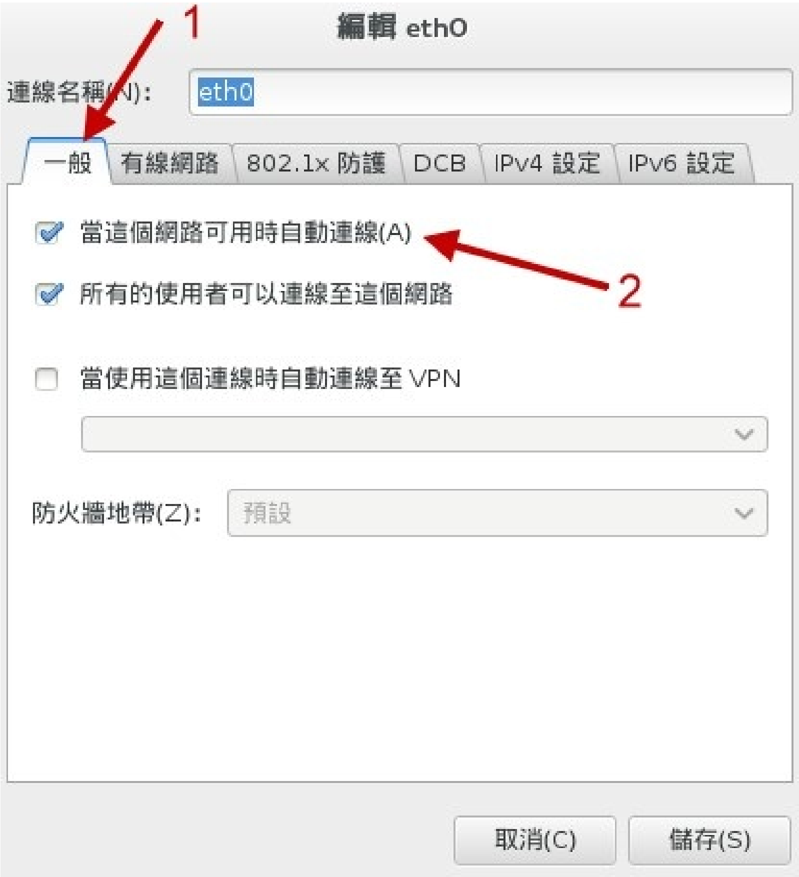


图3.2.35、设置开机自动启动网络

现在 CentOS 7 开机后，默认是没有启动网络的，因此你得要在上图中选择 2 号箭头的“当这个网络可用时自动连线”的项目才行！



图3.2.36、手动设置 IP 的示意图

如上图所示，选择 IPv4 的项目，然后调整 2 号箭头成为手动，接下来按下 3 号箭头加入项目后，才能够在 4 号箭头输入所需要的 IP 位址与网络遮罩～ 写完之后其他的项目不要更动，就按下 5 号箭头的储存吧！然后回到如同下图的画面：

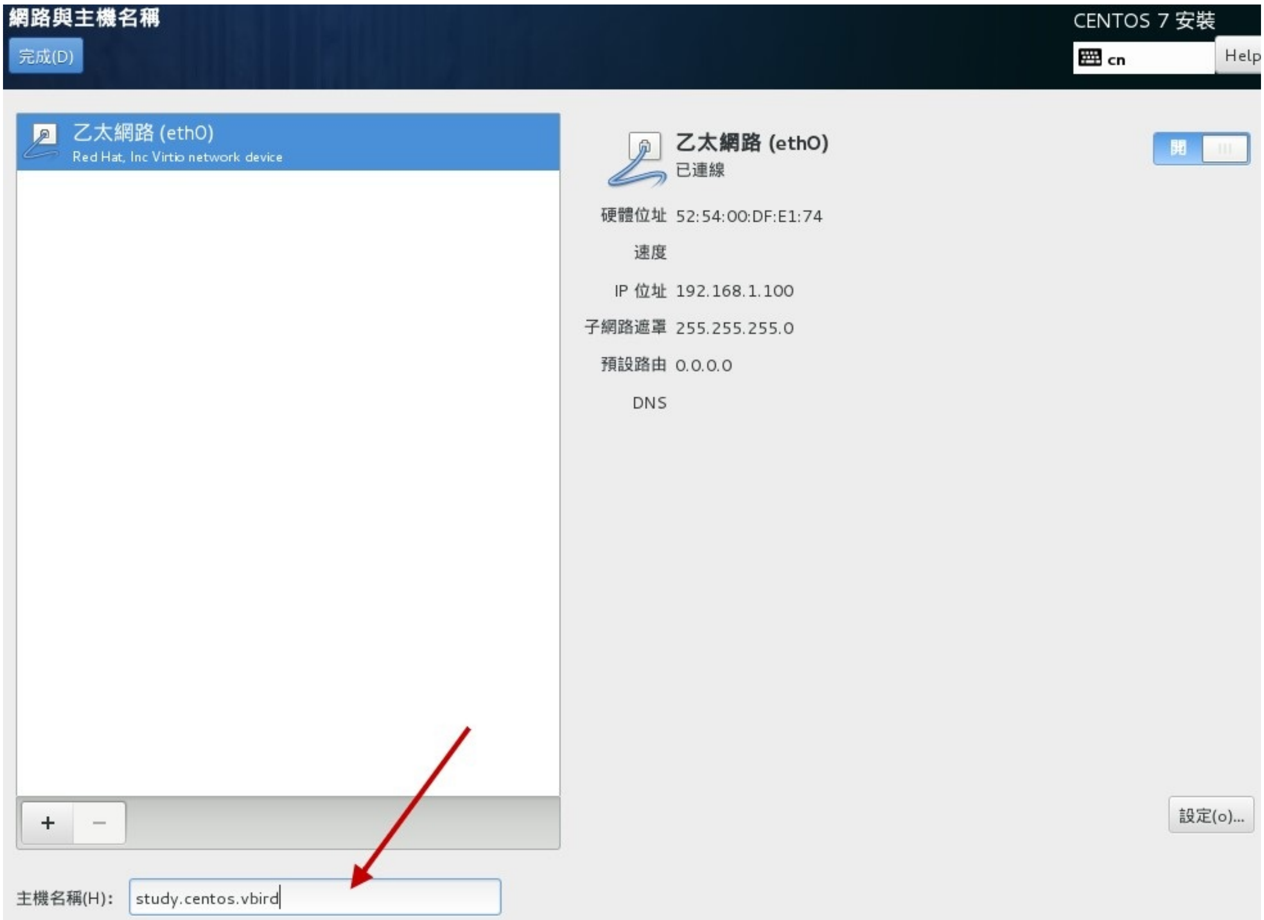


图3.2.37、修改主机名称

如上图所示，右边的网络参数部份已经是正确的了，然后在箭头处输入 3.1 小节谈到的主机名称吧！写完就给它“完成”啰！

3.2.7 开始安装、设置 root 密码与新增可切换身份之一般用户

如果一切顺利的话，那么你应该就可以看到如下的图示，所有的一切都是正常的状态！因此你就可以按下下面图示的箭头部份， 开始安装的流程啰！



图3.2.38、设置完毕并准备开始安装的示意图

现在的安装画面作的还挺简单的，省略了一堆步骤！上述画面按下开始安装后，这时你就可以一边让系统安装， 同时去设置其他项目，可以节省时间啦！如下图所示，还有两件重要的事件要处理，一个是 root 密码，一个是一般身份用户的创建！



图3.2.39、进行安装程序中，还可以持续其他任务的过程

将上图中，按下 ROOT 密码，可以得到下面的图示来修改系统管理员的密码喔！

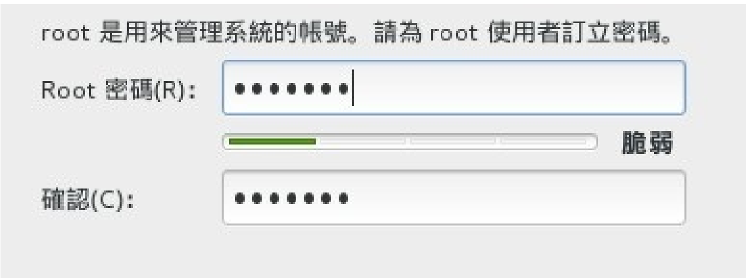


图3.2.40、设置系统管理员 root 的密码

基本上，你可以设置任何密码内容！只是，系统会主动帮你判断你的密码设置的好不好。如果不够好，那么画面中就会告诉你，你的密码很虚弱啦！ 你还是可以坚持你的简易密码！只是，就得要按下两次“完成”，安装程序才会真的帮你设置该密码。

什么是好的密码呢？基本上，密码字符长度设置至少8个字符以上，而且含有特殊符号更好，且不要是个人的可见信息 （如电话号码、身份证、生日等等，就是比较差的密码） 。 例如：I&my_dog之类，有点怪，但是对你又挺好记的密码！就是还 OK 的密码设置喔！



Tips 好的习惯还是从头就开始养成比较好。以前鸟哥上课为了简易的操作，所以给学生操作的系统中，选了个 1234 作为密码，要命了！后来鸟哥的专题生， 实际上线的计算机中，竟然密码还是使用 1234 耶～一上线之后的后果，当然就是被绑架了！还有什么说的？所以，还是一开始就养成好习惯较佳！

管理员密码设置妥当后，接下来鸟哥建议你还是得要创建一个日常登陆系统的惯用一般帐号较好！为什么呢？ 因为通常远端系统管理流程中，我们都会建议将管理员直接登陆的权限拿掉，有需要才用特殊指令 （如 su, sudo 等等，指令后续会谈到！） 切换成管理员身份。 所以啊，你一定得要创建一个一般帐号才好。鸟哥这里使用自己的名子 dmtsai 来作为一个帐号喔！



图3.2.41、创建一个一般帐号

这个帐号既然是你要使用的，那么这个帐号应该就是你认可的管理员使用的一般帐号啊！所以你或许会希望这个帐号可以使用自己的密码来切换身份成为 root， 而不用知道 root 的密码！果真如此的话，那么上头的 2 号箭头处，就得要勾选才好！未来你就可以直接使用 dmtsai 的密码变成 root 哩！ 方便你自己管理～这样即使 root 密码忘记了，你依旧可以切换身份变 root 啊！



图3.2.42、安装完毕的示意图

等到安装妥当之后，你应该就会见到如上的图示！上方的箭头比较有趣！仔细看，你会发现有个"将创建管理员 dmtsai "的项目！ 那就是因为你勾选了"让这位使用者成为管理员"的缘故！当然啦！这个帐号的密码也就很重要！不要随便流出去啊！确定一切事情都顺利搞定，按下箭头处的"重新开机"吧！准备来使用 CentOS Linux 啰！

3.2.8 准备使用系统前的授权同意

重新开机完毕后，系统会进入第一次使用的授权同意画面！如下所示：



图3.2.43、第一次使用 CentOS 7 图形接口的授权同意过程

点选上图中的 1 号箭头后，就会出现如下图所示的授权同意书！

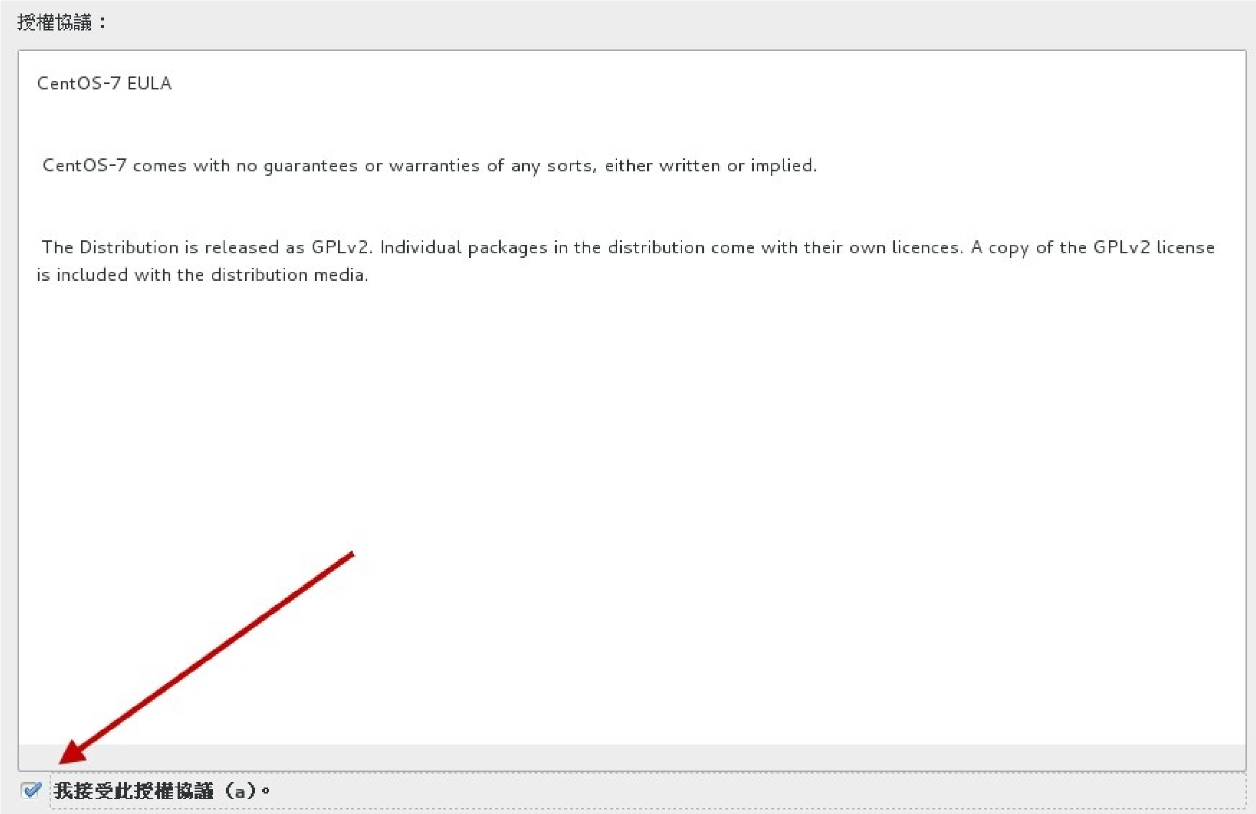
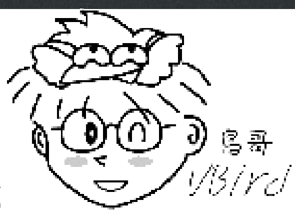


图3.2.44、授权同意书的签署

再次确认后，你就会发现如同下图所示的画面，等待登陆了！第一次登陆系统的相关数据就请看下一个小节啰！



图3.2.45、等待使用者登陆示意图



Tips 先提醒你自己记一下，你刚刚上面所选择的项目，包括 root 的密码等等，通通都会被纪录到 /root/anaconda-ks.cfg 这个文件内喔！ 这个文件可以提醒与协助你未来想要重建一个一模一样的系统时，就可以参考该文件来制作啰！当然，你也可以 google 一下，找 kickstart 这个关键字，会得到很多协助喔！ ^_^

3.2.9 其他功能：RAM testing, 安装笔记本电脑的核心参数（Option）

其实安装光盘还可以进行救援、烧机等任务喔！赶紧来瞧瞧：

- 内存压力测试：memtest86 [3]

CentOS的DVD除了提供一般PC来安装Linux之外，还提供了不少有趣的东西，其中一个就是进行“烧机”的任务！ 这个烧机不是台湾名产烧酒鸡啊，而是当你组装了一部新的个人计算机，想要测试这部主机是否稳定时， 就在这部主机上面运行一些比较耗系统资源的程序，让系统在高负载的情况下去运行一阵子（可能是一天）， 去测试稳定性的一种情况，就称为“烧机”啦！

那要如何进行呢？让我们重新开机并回到图3.2.8 的画面中，然后依序选择“Troubleshooting”、 “Run a memory test”的项目，你的画面就会变成如下的模样了：

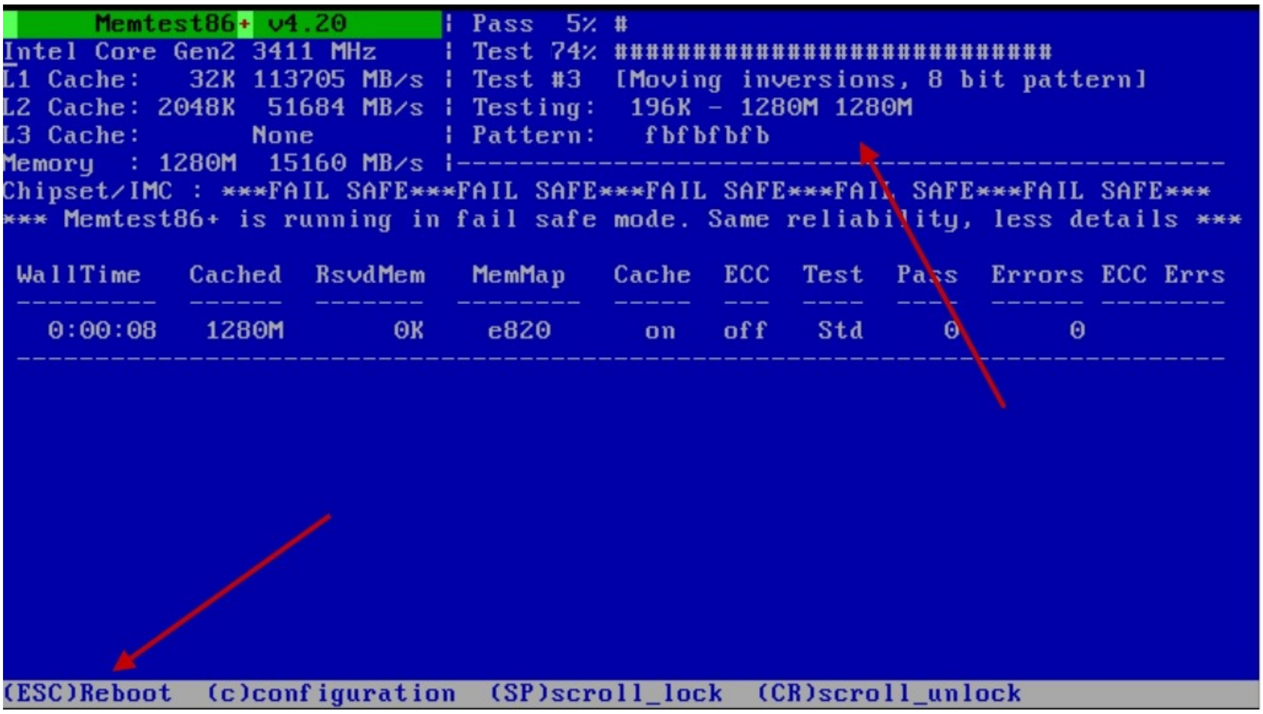


图3.2.46、memory test 的图示

画面中的右上角数据会一直跑，直到你按下 [esc] 按钮为止，他都会一直去操内存！由于内存是服务器当中一个相当重要的元件， 他只要不出事，系统总是稳定的多！所以，通过这个方式来操内存，让内存一直保持在忙碌的状态～等待一天过去，你就可以说，恩！ 这部计算机硬件应该还算稳定吧！ ^_^

- 安装笔记本电脑或其他类PC计算机的参数

由于笔记本电脑加入了非常多的省电机制或者是其他硬件的管理机制，包括显卡常常是整合型的， 因此在笔记本电脑上面的硬件常常与一般桌面电脑不怎么相同。所以当你使用适合于一般桌面电脑的DVD来安装Linux时， 可能常常会出现一些问题，导致无法顺利的安装Linux到你的笔记本电脑中啊！那怎办？

其实很简单，只要在安装的时候，告诉安装程序的linux核心不要载入一些特殊功能即可。 最常使用的方法就是，在使用DVD开机时，选择“”然后按下 [tab] 按键后，加入下面这些选项：

```
nofb apm=off acpi=off pci=noacpi
```

apm（Advanced Power Management）是早期的电源管理模块，acpi（Advanced Configuration and Power Interface）则是近期的电源管理模块。这两者都是硬件本身就有支持的，但是笔记本电脑可能不是使用这些机制， 因此，当安装时启动这些机制将会造成一些错误，导致无法顺利安装。

nofb则是取消显卡上面的缓冲内存侦测。因为笔记本电脑的显卡常常是整合型的， Linux安装程序本身可能就不是很能够侦测到该显卡模块。此时加入nofb将可能使得你的安装过程顺利一些。

对于这些在开机的时候所加入的参数，我们称为“核心参数”，这些核心参数是有意义的！ 如果你对这些核心参数有兴趣的话，可以参考文后的参考数据来查询更多信息[4]。

3.3 多重开机安装流程与管理（Option）

有鉴于自由软件的蓬勃发展以及专利软件越来越贵，所以政府单位也慢慢的希望各部门在选购计算机时， 能够考虑同时含有两种以上操作系统的机器了。加上很多朋友其实也常常有需要两种不同操作系统来处理日常生活与工作的事情。 那我是否需要两部主机来操作不同的操作系统？不需要的，我们可以通过多重开机来选择登陆不同的操作系统喔！ 一部机器搞定不同操作系统哩。



Tips 你可能会问：“既然虚拟机这么热门，应用面也广，那为啥不能安装 Linux 上面使用 windows 虚拟机？或反过来使用呢？” 原因无他，因为“虚拟机在图形显示的性能依旧不足”啊！所以，某些时刻你还是得要使用实体机器去安装不同的操作系统啊！

不过，就如同鸟哥之前提过的，多重开机系统是有很多风险存在的，而且你也不能随时变动这个多重操作系统的开机扇区， 这对于初学者想要“很勐烈的”玩Linux是有点妨碍～所以，鸟哥不是很建议新手使用多重开机啦！ 所以，下面仅是提出一个大概，你可以看一看，未来我们谈到后面的章节时，你自然就会有“豁然开朗”的笑容出现了！ ^_^

3.3.1 安装 CentOS 7.x + windows 7 的规划

由于鸟哥身边没有具有 UEFI BIOS 的机器，加上 Linux 对于 UEFI 的支持还有待持续进步，因此，下面鸟哥是使用虚拟机创建 200GB 的磁盘， 然后使用传统 BIOS 搭配 MBR 分区表来实做多重开机的项目。预计创建 CentOS 7.x 以及一个 Windows 7 的多重操作系统，同时拥有一个共享的数据磁盘。



Tips 为什么要用 MBR 而不用本章之前介绍的 GPT 呢？这是因为“Windows 8.1 以前的版本，不能够在非 UEFI 的 BIOS 环境下使用 GPT 分区表的分区来开机”啊！ 我们既然没有 UEFI 的环境，那自然就无法使用 GPT 分区来安装 Windows 系统了。但其实 windows 还是可以使用 GPT，只是“开机的那颗硬盘，必须要在 MBR 的分区磁盘中”。 例如 C 盘单颗硬盘使用 MBR ，而数据磁盘 D 盘使用 GPT ，那就 OK 没问题！

另外，与过去传统安装流程不同，这次鸟哥希望保留 Linux （因为开机管理是由 Linux 管的）在前面，windows 在后面的分区内， 因此需要先安装 Linux 后再安装 windows，后来通过修改系统配置文件来让系统达成多重开机！基本上鸟哥的分区是这样规划的 （因为不用 GPT，所以无须 BIOS Boot 项目）：

Linux设备文件名	Linux载点	Windows设备	实际内容	文件系统	容量
/dev/vda1	/boot	-	Linux 开机信息	xfs	2GB
/dev/vda2	/	-	Linux 根目录	xfs	50GB
/dev/vda3	-	C	Windows 系统盘	NTFS	100GB
/dev/vda5	/data	D	共享数据磁盘	VFAT	其他剩余

再次强调，我们得要先安装 Linux 在通过后续维护的方案来处理的喔！而且，为了强制 Windows 要安装在我们要求的分区， 所以在 Linux 安装时，得要将上述的所有分区先分区出来喔！大概就是这样！来实作吧！

3.3.2 进阶安装 CentOS 7.x 与 Windows 7

请依据本章前面的方式一项一项来进行各项安装行为，比较需要注意的地方就是安装时，不可以加上 inst.gpt 喔！我们单纯使用 MBR 分区啊！

进行到 图 3.2.12 的项目时，先不要选择分区，请按下“ [ctrl]+[alt]+[f2] ”来进入安装过程的 shell 环境。 然后进行如下的动作来预先处理好你的分区！因为鸟哥使用图形化界面的分区模式，老是没有办法调出满意的顺序！ 只好通过如下的手动方式来创建啰！但是你得要了解 parted 这个指令才行！

```
[anaconda root@localhost /]# parted /dev/vda mklabel msdos # 创建 MBR 分区
[anaconda root@localhost /]# parted /dev/vda mkpart primary 1M 2G # 创建 /boot
[anaconda root@localhost /]# parted /dev/vda mkpart primary 2G 52G # 创建 /
[anaconda root@localhost /]# parted /dev/vda mkpart primary 52G 152G # 创建 C
[anaconda root@localhost /]# parted /dev/vda mkpart extended 152G 100%# 创建延伸分区
[anaconda root@localhost /]# parted /dev/vda mkpart logical 152G 100% # 创建逻辑分区
[anaconda root@localhost /]# parted /dev/vda print # 显示分区结果
```

如果按照上面的处理流程，由于原本是 MBR 的分区，因此经过 mklabel 的工作，将 MBR 强制改为 GPT 后， 所有的分区就死光光了！因此不用删除就不会有剩余。接下来就是创建五个分区，最终的 print 行为就是列出分区结果， 结果应该有点像下面这样：

```
[anaconda root@localhost /]# parted /dev/vda print
Model: Virtio Block Device (virtblk)
Disk /dev/vda: 215GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number  Start   End     Size    Type      File system  Flags
 1      1049kB  2000MB  1999MB  primary
 2      2000MB  52.0GB  50.0GB  primary
 3      52.0GB  152GB   100GB   primary
 4      152GB   215GB   62.7GB  extended               lba
 5      152GB   215GB   62.7GB  logical
```

图3.3.1、本范例的分区结果

接下来再次按下“ [ctrl]+[alt]+[f6] ”来回到原本的安装流程中，然后一步一步实做到分区区那边， 然后依据相关的设备文件名来进行“重新格式化”并填入正确的挂载点，最终结果有点像下面这样：



图 3.3.2、安装流程的分区情况

你会看到有个“重新格式化”的项目吧！那个一定要勾选喔！之后就给它持续的安装下去，直到装好为止喔！安装完毕之后， 你也无须进入到设置的项目，在重新开机后，塞入 windows 7 的原版光盘，之后持续的安装下去！要注意，得要选择那个 100G 容量的分区安装才行！ 最重要的那个安装画面有点像下面这样：



图 3.3.3、安装 windows 的分区示意图

一样，让 windows 自己安装到完毕吧！

3.3.3 救援 MBR 内的开机管理程序与设置多重开机菜单

为了应付分区工作，所以我们是先安装 Linux 再安装 Windows 的。只是，如此一来，整颗硬盘的 MBR 部份就会被 windows 的开机管理程序占用了！ 因此，安装好了 Windows 的现在，我们得要开始来救援 MBR，同时编辑一下开机菜单才行！

- 救援回 Linux 的开机管理程序：

救援 Linux 开机管理程序也不难，首先，放入原版光盘，重新开机并且进入类似 图 3.2.8 的画面中，然后依据下面的方式来处理救援模式。 进入“ Troubleshooting ”，选择“ Rescue a CentOS system ”，等待几秒钟的开机过程，之后系统会出现如下的画面，请选择“ Continue ”喔！

图3.3.4、如何使用找到的 Linux 磁盘系统，建议用 Continue （RW） 模式

如果真的有找到 Linux 的操作系统，那么就会出现如下的图示，告诉你，你的原本的系统放置于 /mnt/sysimage 当中喔！



图3.3.5、找到了 CentOS 操作系统时，可以进行任务了

接着下来准备要救援 MBR 的开机管理程序啰！处理的方法指令如下：

```
sh-4.2# chroot /mnt/sysimage
sh-4.2# grub2-install /dev/vda
Installing for i386-pc platform.
Installation finished. No error reported.
sh-4.2# exit
sh-4.2# reboot
```

- 修改开机菜单任务：

接下来我们可以修订开机菜单了！不然开机还是仅有 Linux 而已～先以正常流程登陆 Linux 系统，切换身份成为 root 之后，开始进行下面的任务：

```
[root@study ~]# vim /etc/grub.d/40_custom
#!/bin/sh
exec tail -n +3 $0
# This file provides an easy way to add custom menu entries.  Simply type the
# menu entries you want to add after this comment.  Be careful not to change
# the 'exec tail' line above.
menuentry "Windows 7" {
   set root='（hd0,3）'
   chainloader +1
}

[root@study ~]# vim /etc/default/grub
GRUB_TIMEOUT=30  # 将 5 秒改成 30 秒长一些
...
[root@study ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

接下来就可以测试能否成功了！如果一切顺利的话，理论上就能够看到如下的图示，并且可以顺利的进入 Linux 或 Windows 啰！加油！

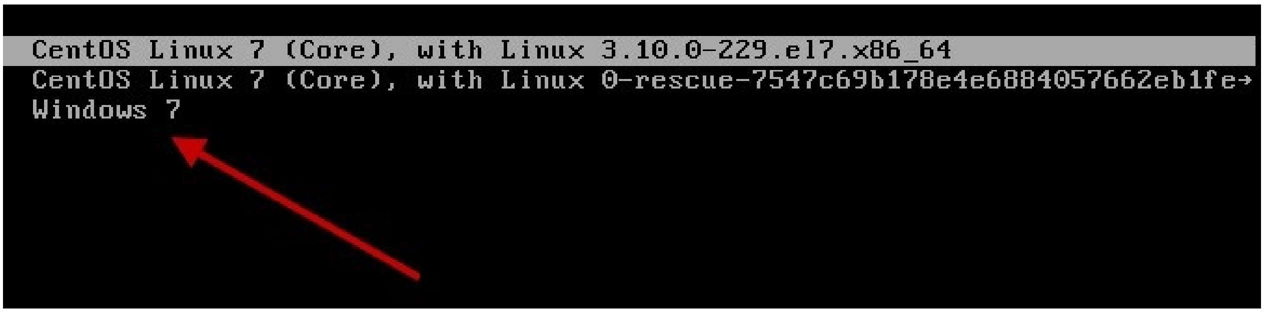


图3.3.6、多重开机的开机菜单示意

- 后续维护的注意事项

多重开机设置完毕后请特别注意，（1）Windows的环境中最好将Linux的根目录与swap取消挂载，否则未来你打开文件资源管理器时，该软件会要求你“格式化！”如果一个不留神，你的Linux系统就毁了。（2）你的Linux不可以随便的删除！因为grub会去读取Linux根目录下的/boot/目录内容，如果你将Linux移除了，你的Windows也就无法开机了！因为整个开机菜单都会不见喔！

3.4 重点回顾

- 不论你要安装什么样的Linux操作系统角色，都应该要事先规划例如分区、开机管理程序等；
- 建议练习机安装时的磁盘分区能有/, /boot, /home, swap四个分区；
- 安装CentOS 7.x的模式至少有两种，分别是图形接口与命令行；
- CentOS 7 会主动依据你的磁盘容量判断要用 MBR 或 GPT 分区方式，你也可以强迫使用 GPT；
- 若安装笔记本电脑时失败，可尝试在开机时加入“linux nofb apm=off acpi=off”来关闭省电功能；
- 安装过程进入分区后，请以“自订的分区模式”来处理自己规划的分区方式；
- 在安装的过程中，可以创建逻辑卷轴管理员 （LVM）；
- 一般要求swap应该要是1.5~2倍的实体内存量，但即使没有swap依旧能够安装与运行Linux操作系统；
- CentOS 7默认使用 xfs 作为文件系统
- 没有连上Internet时，可尝试关闭防火墙，但SELinux最好选择“强制”状态；
- 设置时不要选择启动kdump，因为那是给核心开发者查阅死机数据的；
- 可加入时间服务器来同步化时间，台湾可选择tock.stdtime.gov.tw这一部；
- 尽量使用一般用户来操作Linux，有必要再转身份成为root即可。
- 即使是练习机，在创建 root 密码时，建议依旧能够保持良好的密码规则，不要随便设置！

3.5 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ） 问答题部分：

- Linux的目录配置以“树状目录”来配置，至于磁盘分区（partition）则需要与树状目录相配合！ 请问，在默认的情况下，在安装的时候系统会要求你一定要分区出来的两个Partition为何？就是根目录“/”与内存交换空间“Swap”
- 默认使用 MBR 分区方式的情况下，在第二颗 SATA 磁盘中，分区“六个有用”的分区 （具有 filesystem 的） ，此外，已知有两个 primary 的分区类型！请问六个分区的文件名？/dev/sdb1（primary） /dev/sdb2（primary） /dev/sdb3（extended） /dev/sdb5（logical 下面皆为 logical） /dev/sdb6 /dev/sdb7 /dev/sdb8 请注意，5-8 这四个 logical 容量相加的总和为 /dev/sdb3！
- 什么是GMT时间？台北时间差几个钟头？GMT 时间指的是格林威治时间，称为标准的时间，而台北时间较 GMT 快了 8 小时！
- 软件磁盘阵列的设备文件名为何？RAID : /dev/md[0-127];
- 如果我的磁盘分区时使用 MBR 方式，且设置了四个 Primary 分区，但是磁盘还有空间，请问我还能不能使用这些空间？不行！因为最多只有四个 Primary 的磁盘分区，没有多的可以进行分区了！且由于没有 Extended ，所以自然不能再使用 Logical 分区

3.6 参考资料与延伸阅读

- [1]虚拟机管理员创建一部虚拟机的流程： http://www.cyberciti.biz/faq/kvm-virt-manager-install-centos-linux-guest/ http://www.itzgeek.com/how-tos/linux/centos-how-tos/install-kvm-qemu-on-centos-7-rhel-7.html#axzz3Yf6il9S2 https://virt-manager.org/screenshots/
- [2]CentOS 7 网卡的命名规则： https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Networking_Guide/sec-Understanding_the_Predictable_Network_Interface_Device_Names.html
- [3]进阶内存测试网站：http://www.memtest.org/
- [4]更多的核心参数可以参考如下链接： http://www.faqs.org/docs/Linux-HOWTO/BootPrompt-HOWTO.html 对于安装过程所加入的参数有兴趣的，则可以参考下面这篇链接，里面有详细说明硬件原因： http://polishlinux.org/choose/laptop/
- 安装过程的简易示意图： http://www.tecmint.com/centos-7-installation/ https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/html/Installation_Guide/sect-disk-partitioning-setup-x86.html

2015/05/06：首次释出新版本

第四章、首次登陆与线上求助

最近更新日期：20//

终于可以开始使用Linux这个有趣的系统了！由于Linux系统使用了非同步的磁盘/内存数据传输模式，同时又是个多用户多任务的环境， 所以你不能随便的不正常关机，关机有一定的程序喔！错误的关机方法可能会造成磁盘数据的损毁呢！ 此外，Linux有多种不同的操作方式，图形接口与命令行的操作有何不同？ 我们能否在命令行取得大量的指令说明，而不需要硬背某些指令的选项与参数等等。这都是这一章要来介绍的呢！

4.1 首次登陆系统

登陆系统有这么难吗？并不难啊！虽然说是这样说，然而很多人第一次登陆Linux的感觉都是“接下来我要干啥？”如果是以图形接口登陆的话，或许还有很多好玩的事物， 但要是以命令行登陆的话，面对着一片黑压压的屏幕，还真不晓得要干嘛呢！为了让大家更了解如何正确的使用Linux， 正确的登陆与离开系统还是需要说明的！

4.1.1 首次登陆CentOS 7.x图形接口

开机就开机呀！怎么还有所谓的登陆与离开呀？不是开机就能够用计算机了吗？ 开什么玩笑，在Linux系统中由于是多用户多任务的环境，所以系统随时都有很多不同的用户所下达的任务在进行， 因此正确的开关机可是很重要的！不正常的关机可能会导致文件系统错乱，造成数据的毁损呢！这也是为什么通常我们的Linux主机都会加挂一个不断电系统啰！

如果在第三章一切都顺利的将CentOS 7.x完成安装并且重新开机后， 应该就会出现如下的等待登陆的图形画面才对。画面中1号箭头显示目前的日期与时间，2号箭头则是辅助功能、语系、音量与关机钮， 3号箭头就是我们可以使用帐号登陆的输入框框，至于4号箭头则是在使用特别的帐号登陆时才会用到的按钮。

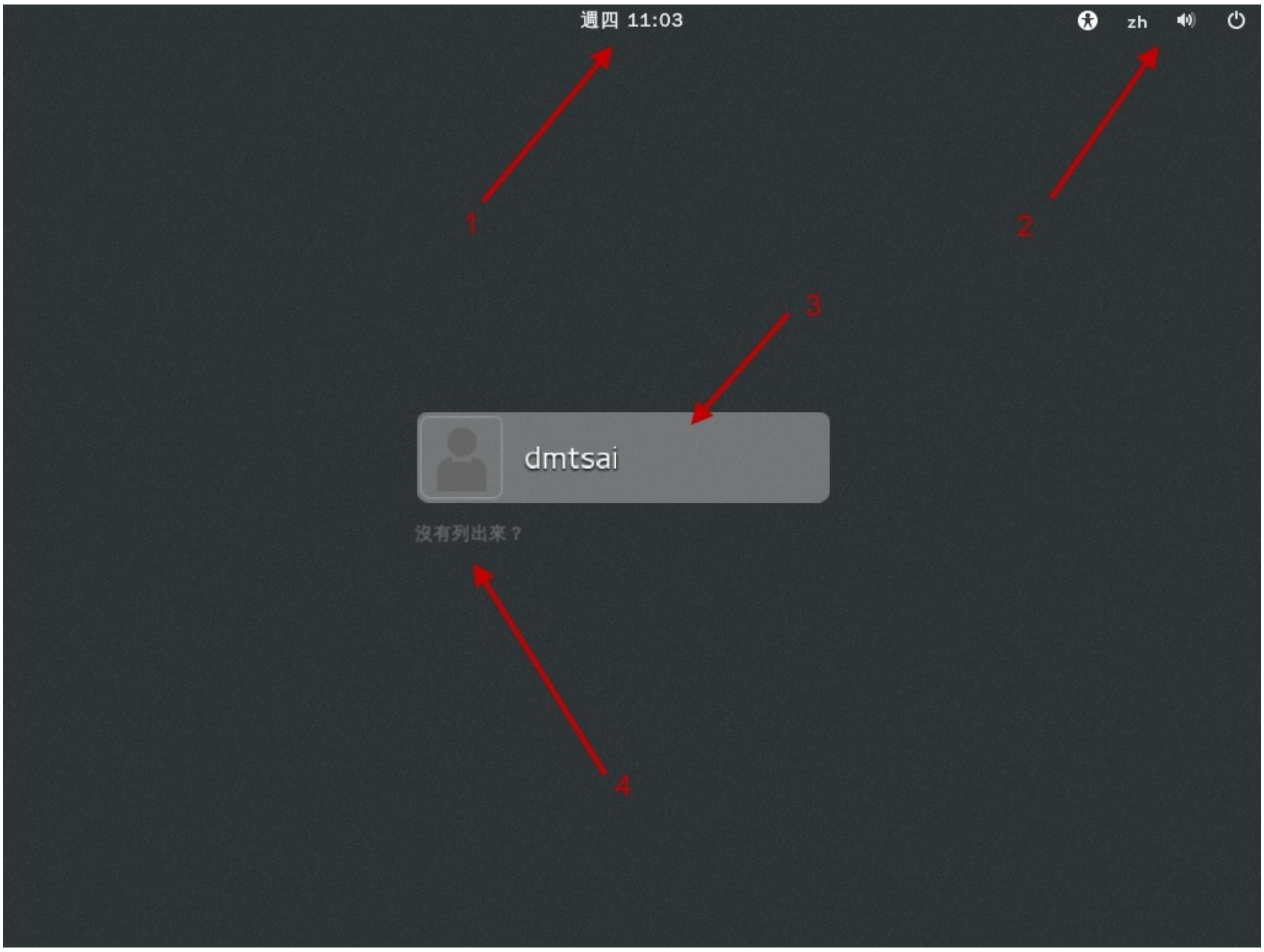


图4.1.1、X等待登陆的画面示意图

接下来让我们来了解一下这个登陆画面的相关功能吧！首先，在箭头 1 的地方，如果你动鼠标过去点一下，就会出现如下的窗口， 主要在告诉你日期、日历与时间而已～如下图所示，鸟哥撷取这张图的时间就是在 2015/05/21 早上喔！



图4.1.2、X等待登陆的画面示意图-日历、时间显示

然后看一下右上角的角落，你会发现有个小人形图示，那个是协助登陆的无障碍画面处理！如果你的键盘暂时出了点问题，某些按键无法按， 那就可以使用如下画面的“屏幕键盘”的项目，将他 On 一下～那未来有需要在登陆的时候有打字的需求时，屏幕就会出现类似手机要你打字的键盘画面啦！



图4.1.3、X等待登陆的画面示意图-无障碍登陆协助

有看到那个 zh 嘛？那个是语系的选择～点下去你会看到这部系统支持的语系数据有多少。至于那个类似喇叭的小图示，就是代表着音效的大小声控制～ 而最右边那个有点像是关机的小图示又是干麻的呢？没关系！别紧张！用力点下去看看～就会出现如下图示，其实就是准备要关机的一些功能按钮～ 暂停是进入休眠模式，重新启动就是重新开机啊，关闭电源当然就是关机啰！所以，你不需要登陆系统，也能够通过这个画面来“关机”喔！



图4.1.4、X等待登陆的画面示意图-无须登陆的关机与重新开机

接下来看到图4.1.1的地方，图示中的箭头 3,4 指的地方就是可以登陆的帐号！一般来说，能够让你输入帐密的正常帐号， 都会出现在这个画面当中，所以列表的情况可能会非常长！那有些特殊帐号，例如我们在第三章安装过程中，曾经有创建过两个帐号， 一个是 root 一个是 dmtsai，那个 dmtsai 可以列出来没问题，但是 root 因为身份比较特殊，所以就没有被列出来！因此， 如果你想要使用 root 的身份来登陆，就得要点选箭头 4 的地方，然后分别输入帐密即可！

如果是一般可登陆正常使用的帐号，如画面中的 dmtsai 的话，那你就直接点选该帐号，然后输入密码即可开始使用我们的系统了！ 使用 dmtsai 帐号来输入密码的画面示意如下：



图4.1.5、X等待登陆的画面示意图-一般帐号登陆系统的密码字段

在你输入正确的密码之后，按下“登陆”按钮，就可以进入 Linux 的图形画面中，并开始准备操作系统啰！



Tips 一般来说，我们不建议您直接使用 root 的身份登陆系统喔！请使用一般帐号登陆！等到有需要修改或者是创建系统相关的管理工作时， 才切换身份成为 root！为什么呢？因为系统管理员的权限太高了！而 Linux 下面很多的指令行为是“没有办法复原”的！所以， 使用一般帐号时，“手滑”的灾情会比较不严重！

4.1.2 GNOME的操作与登出

在每一个用户“第一次”以图形接口登陆系统时，系统都会询问使用者的操作环境，以依据使用者的国籍、 语言与区域等制定与系统默认值不同的环境。如下所示，第一个问题就是询问你未来整体的环境要使用的语系为哪个语系与国家？ 当然我们台湾都选汉语台湾啊 （安装的时候选择的默认值），如果有不同的选择，请自行挑选你想要的环境，然后按下“下一步”即可。



图4.1.6、每个用户第一次登陆系统的环境设置

再来则是选择输入法，除非你有特殊需求，否则不需要修改设置值。若是需要有其他不同的输入法，请看下图左侧箭头指的“+”符号， 按下它就可以开始选择其他的输入法了。一切顺利的话，请点选“下一步”。



图4.1.7、每个用户第一次登陆系统的环境设置

上述的环境选择妥当之后，系统会出现一个确认的画面，然后就出现“入门信息”的类似网页的画面来给你瞧一瞧如何快速入门啰！如下所示。 如果你有需要，请一个一个链接去点选查阅，如果已经知道这是啥东西，也可以如画面箭头处，直接关闭即可！

图4.1.8、每个用户第一次登陆系统的环境设置



Tips 要注意喔！上述的画面其实是 GNOME 的求助软件窗口，并不是浏览器窗口！第一次接触到这个画面的学生， 直接在类似网址列的框框中写入 URL 网址，结果当然是找不到数据...当学生问鸟哥时，鸟哥也被唬住了...以为是浏览器...

终于给他看到图形接口啦！真是很开心吧！如下图所示，整个GNOME的窗口大约分为三个部分：

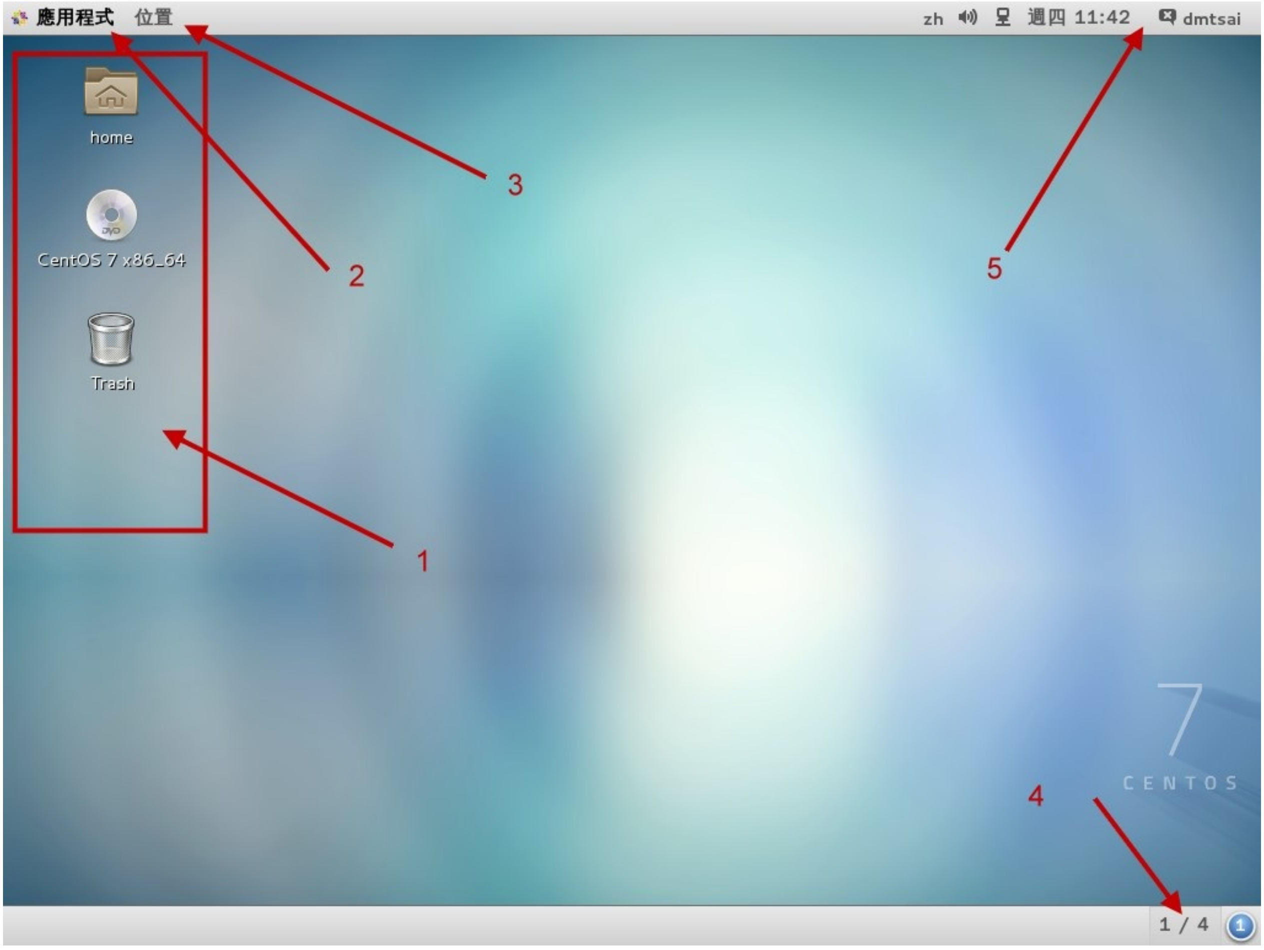


图4.1.9、窗口接口的环境介绍

- 上方工作列（control panel） 上半部左侧有“应用程序”与“位置”，右侧则有“输入法切换”、声音、网络、日期、帐号相关设置切换等，这个位置可以看成是工作列。 举例来说，你可以使用鼠标在 2 号箭头处 （应用程序） 点击一下，就会有更多的程序集出现！然后移动鼠标就能够使用各个软件了。 至于 5 号箭头所指的地方，就是系统时间与声音调整。最右上角则是目前登陆的帐号身份，可以取得很多的设置信息的！

- 桌面 整个画面中央就是桌面啦！在桌面上默认有两个小按钮，例如箭头 1 所指的地方，常见的就是目前这个帐号的主文件夹，你可以使用鼠标连击两下就能够打开该功能。 另一个则是垃圾桶 （Trash）。如果你的安装光盘没有退出，那么该光盘以及其他可能的可携式 USB 设备，也可能显示在桌面上！ 例如图中的“ CentOS 7 x86_64 ”的光片图示，就是你没有退出的光盘喔！

- 下方工作列 下方工作列的目的是将各工作显示在这里，可以方便使用者快速的在各个工作间切换喔！另外，我们还有多个可用的虚拟桌面 （Virtual Desktop），就是画面中右下角那个 1/4 的东东！ 该数字代表的意思是，共有 4 个虚拟桌面，目前在第一个的意思。你可以点一下该处，就知道那是啥东西了！

Linux 桌面的使用方法几乎跟 Windows 一模一样，你可以在桌面上按下右键就可以有额外的菜单出现；你也可以直接按下桌面上的“个人数据夹 （home）”， 就会出现类似Windows 的“文件资源管理器”的文件/目录管理窗口，里面则出现你自己的主文件夹；下面我们就来谈

谈几个在图形接口里面经常使用的功能与特色吧！



Tips 关于“个人数据夹”的内容，记得我们之前说过Linux是多用户多任务的操作系统吧？ 每个人都会有自己的“工作目录”，这个目录是使用者可以完全掌控的， 所以就称为“使用者个人主文件夹”了。一般来说，主文件夹都在/home下面， 以鸟哥这次的登陆为例，我的帐号是dmtsai，那么我的主文件夹就应该在/home/dmtsai/啰！

- 上方工具列：应用程序 （Applications）

让我们点击一下“应用程序”那个按钮吧！看看下拉式菜单中有什么软件可用！如下图所示。



图4.1.10、应用程序集当中，需要注意有阶层的显示喔！

你要注意的是，这一版的 CentOS 在这个应用程序的设计上，阶层式变化间并没有颜色的区分，左侧也没有深色三角形的示意小图，因此如上图所示，如果你想要打开计算机软件， 那得先在左边第一层先移动到“附属应用”之后，鼠标水平横向移动到右边，才可以点选计算机

喔！鸟哥一开始在这里确实容易将鼠标垂直向乱移动，导致老是没办法移动到正确的按钮上！

基本上，这个“应用程序”按钮已经将大部分的软件功能分类了，你可以在里头找到你常用的软件来操作。例如想要使用 Office 的办公室软件， 就到“办公”选项上，就可以看到许多软件存在了！此外，你还会看到最下面有个“活动总览”，那个并没有任何分类的子项目在内，那是啥东西？ 没关系，基本上练习机你怎么玩都没关系！所以，这时就给他点点看啊！会像下面的图示这样：

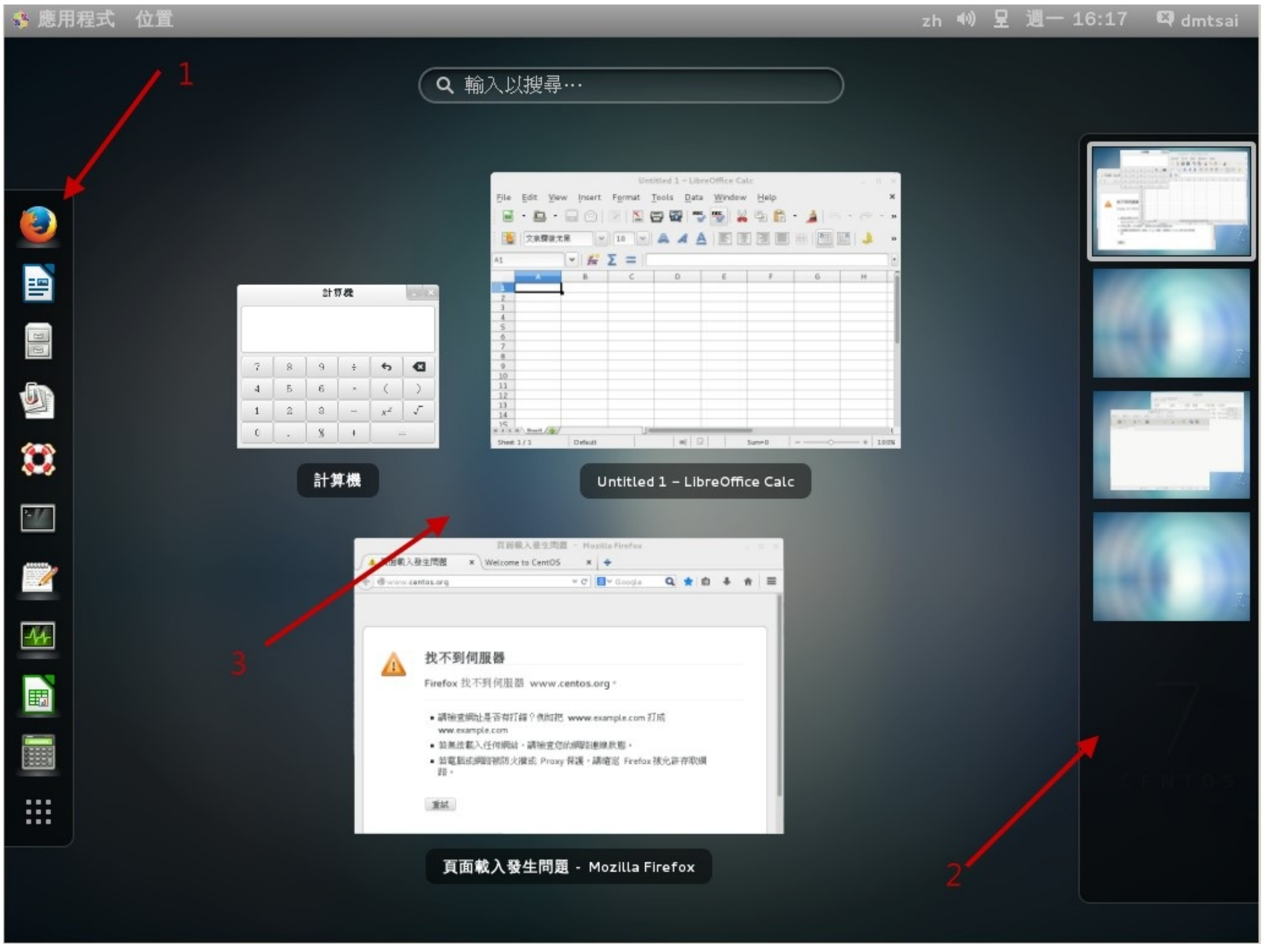


图4.1.11、应用程序的总览画面示意图！

画面左侧 1 号箭头处，其实就是类似快速按钮的地方，可以让你快速的选择你所常用的软件。右侧 2 号箭头处，就是刚刚我们上面谈到的虚拟桌面啰！ 共有四个，而目前画面中显示的最是最上面那个一号桌面的意思。如果细看该区块，就会发现其实鸟哥在第三个虚拟桌面当中也有打开几个软件在操作呢！ 有没有发现啊？至于画面中的 3 号箭头处，就是目前这个活动中的虚拟桌面上，拥有的几个启动的软件啰！你可以点选任何你想要的软件，就可以开始操作该软件了！ 所以使用这个“活动总览”，比较可以让你在开好多窗口的环境下，快速的回到你所需要的软件功能中喔！

- 上方工具列：位置 （就是文件资源管理器）

如果你想要知道系统上面还有哪些文件数据，以及你目前这个帐号的基本子目录，那就得要打开文件资源管理器啰 （file manager）！ 打开文件资源管理器很简单，就是选择左上方那个“位置”的按钮项目即可。在这个项目中主要有几个细项可以直接打开目录的内容，主文件夹、下载、图片、影片等等， 其实除了主文件夹之外，下面的次目录“就是主文件夹下的次目录”啦！所以你可以直接打开主文件夹即可！如下所示：

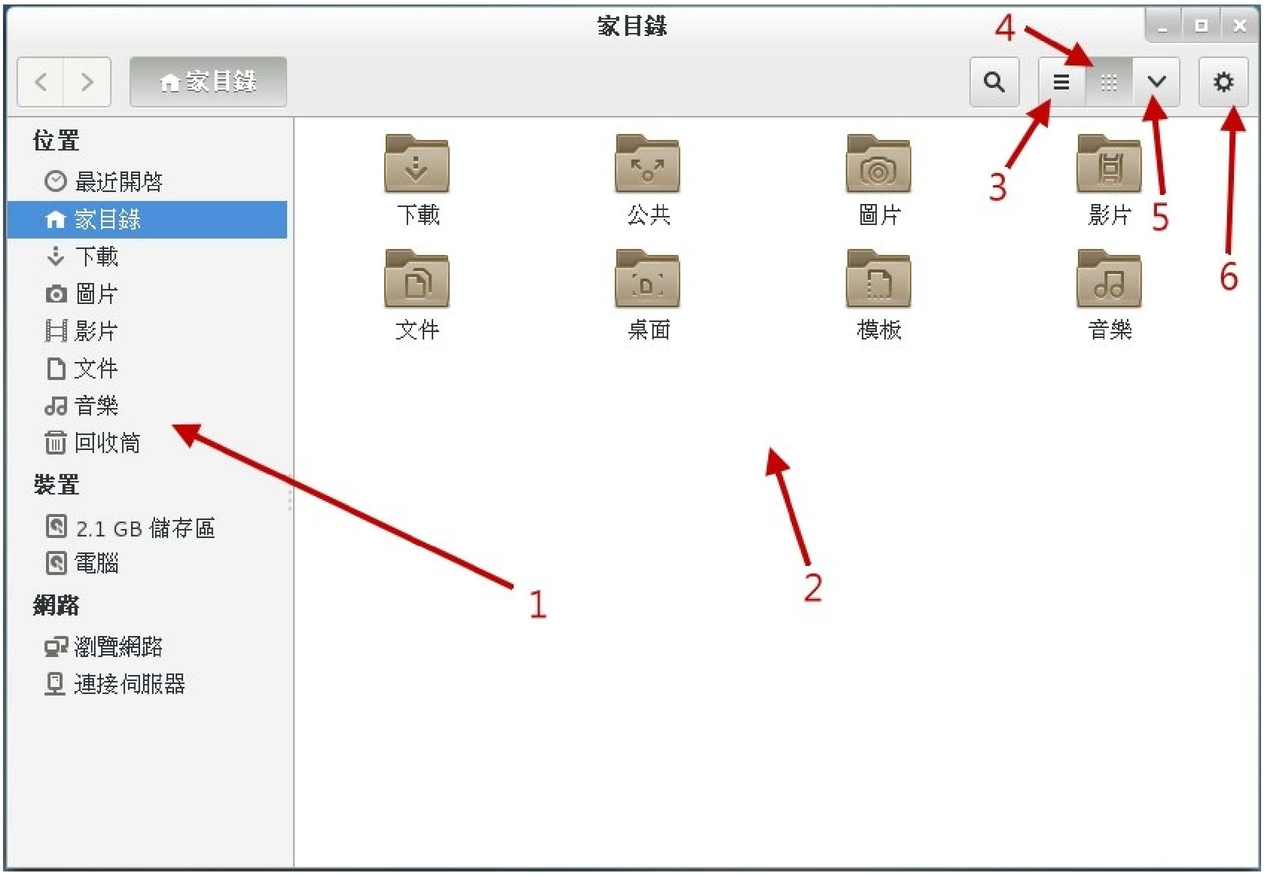


图4.1.12、文件资源管理器操作示意图

如上图所示，1 号箭头处可以让你选择不同的目录或数据来源，2 号箭头则以小图示的方式显示该物件可能是什么数据，3 号箭头则可以将目前的小图示变成详细数据清单， 4 号箭头就是目前小图示的显示模式，5 号箭头可以进行图示数据的放大、缩小、排序方式、是否显示隐藏文件等重要功能！6 号箭头则是其他额外的功能项目！ 好了，线再让我们来操作一下这个软件吧！如果你想要观察每个文件名的详细数据，并且显示“隐藏文件”的话，那该如何处理呢？如下图所示的方式处理一下：

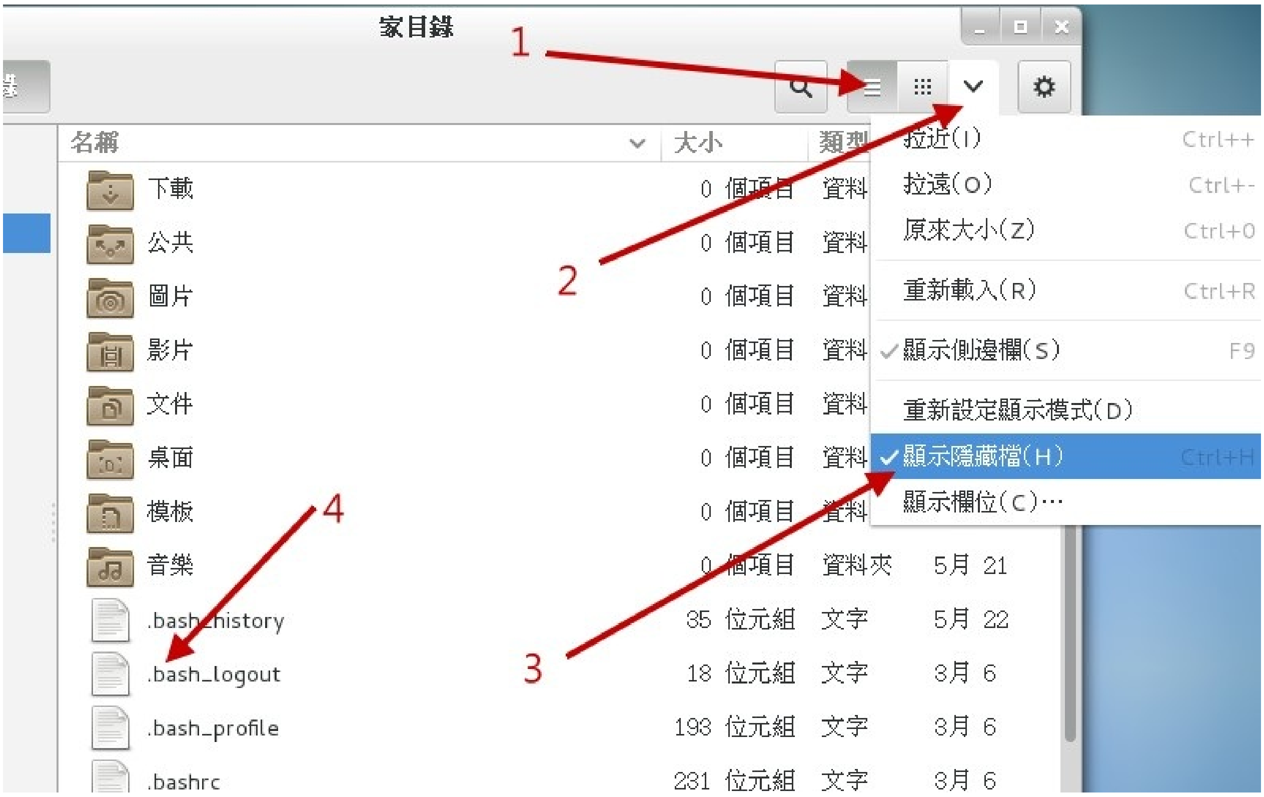


图4.1.13、文件资源管理器操作示意图

按照上面的三个步骤点选完毕后，你就会看到如 4 号箭头处指的，有一些额外的文件名跑出来了！而且，这些跑出来的文件名共同的特色就是“文件名前面开头是小数点 .” 没错！你答对了～只要文件名的开头是由小数点开始的，那么该文件名就不会在一般观察模式被显示出来！所以说，在 Linux 下面，隐藏文件并不是什么特殊的权限， 单纯是因为文件名命名的处理方式来搞定的！这样理解否？

如果你想要观察系统有多少不同的文件系统呢？那就看一下文件资源管理器左侧“设备”的项目下，有几个项目就是有几个设备啰！现在让我们来观察一下“计算机”内有什么数据吧！ 请按下他！然后观察一下如下的图示：

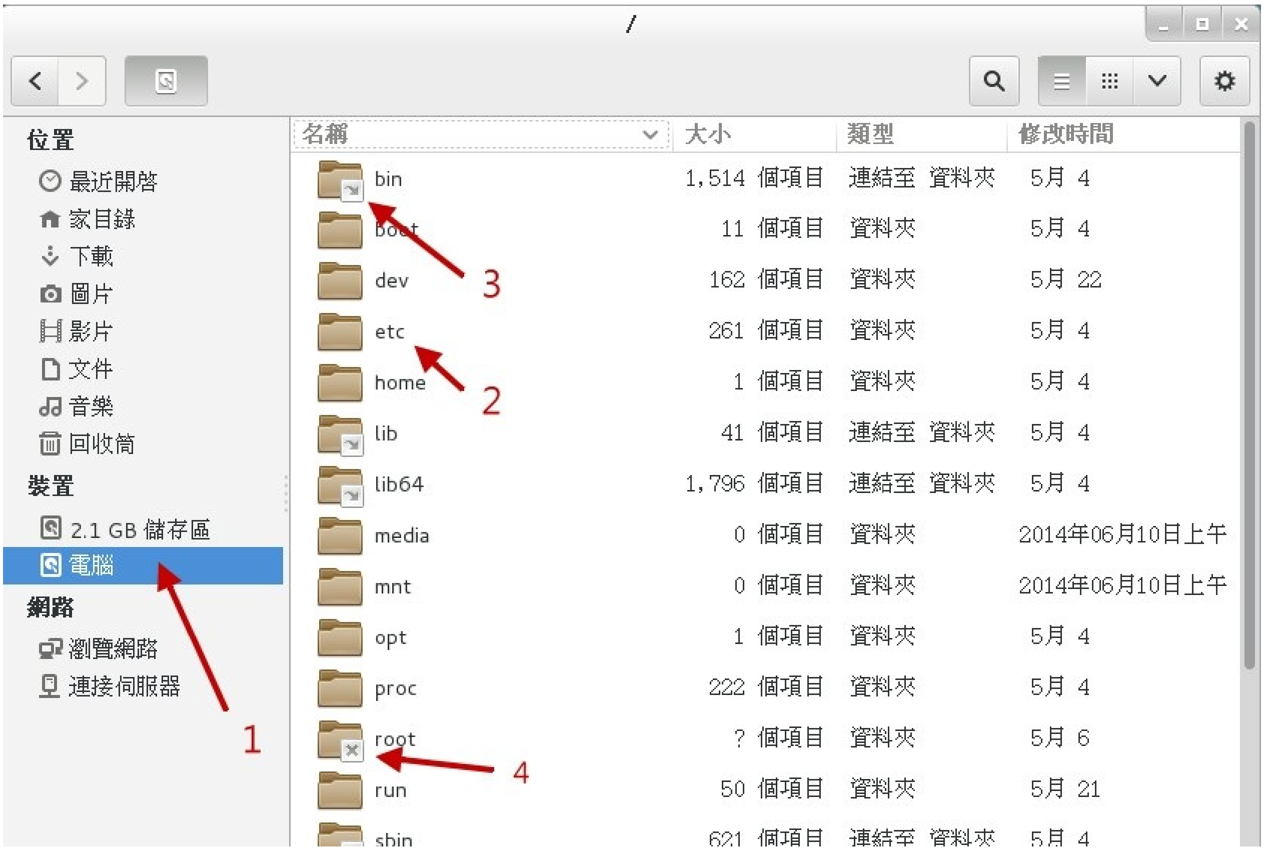


图4.1.14、文件资源管理器操作示意图

如上图所示，点下 1 号箭头后，右边就出现一堆目录数据夹。注意看，2 号箭头处指的是正常的一般目录，3 号箭头则指的是有“链接文件”的数据，这个链接文件可以想像成 Windows 的“捷径”功能就是了～如果你的帐号没有权限进入该目录时，该目录就会出现一个 X 的符号，如同 4 号箭头处！很清楚吧！ 好！让我们来观察一下有没有 /etc -> sysconfig -> network-scripts 这个目录下的数据呢？



图4.1.15、文件资源管理器操作示意图

如果你可以依序双击每个正确的目录，就可以得到如上图示。画面中的 1 号箭头处，可以让你“回到上一个画面”中，不是回到上一层～而是“上一个画面”喔！ 这点要注意。至于 2 号区块处，你可以发现有不同颜色的显示，最右边的是目前所在目录，所以 3 号画面就显示该目录下的文件信息。 你可以快速的点选 2 号区块处的任何一个目录，就可以快速的回到该层目录中去查看文件数据喔！

- 中文输入法与设置

如果你在安装的时候就选定中文，并且有处理过切换中/英文的快速键，那这个项目几乎可以不用理他了！但是如果你都使用默认值来安装时， 可能会发生没办法使用惯用的“ctrl+shift”或“ctrl+space”来切换中文的问题！同时，也可能没办法找到你想要的中文输入法～那怎办？没关系， 请使用图4.1.9画面中右上角的帐号名称处点一下，然后选择“设置值”，或者从“应用程序”、“系统工具”、“设置值”也可以打开它！ 之后选择“地区和语言”项目，就可以得到如下画面。



图4.1.16、地区与语言设置项目

在上面的画面中，你可以按下箭头所指的地方，就可以增加或减少输入法的项目了。但是，如果想要切换不同的语言呢？那请回到原本的设置画面， 之后请选择“键盘”的项目，并按下“快捷键”，出现如下的画面，点选在画面中的左侧“输入”项目，并在“切换到下一个输入来源”点选一、两下， 等到出现如 3 号箭头处出现“新捷径键”时，按下你所需要的组合键，例如鸟哥习惯按“crtl + space”，那就自己按下组合键， 之后你就可以使用自己习惯的输入法切换快速键，来变更你所需要的输入法啰！

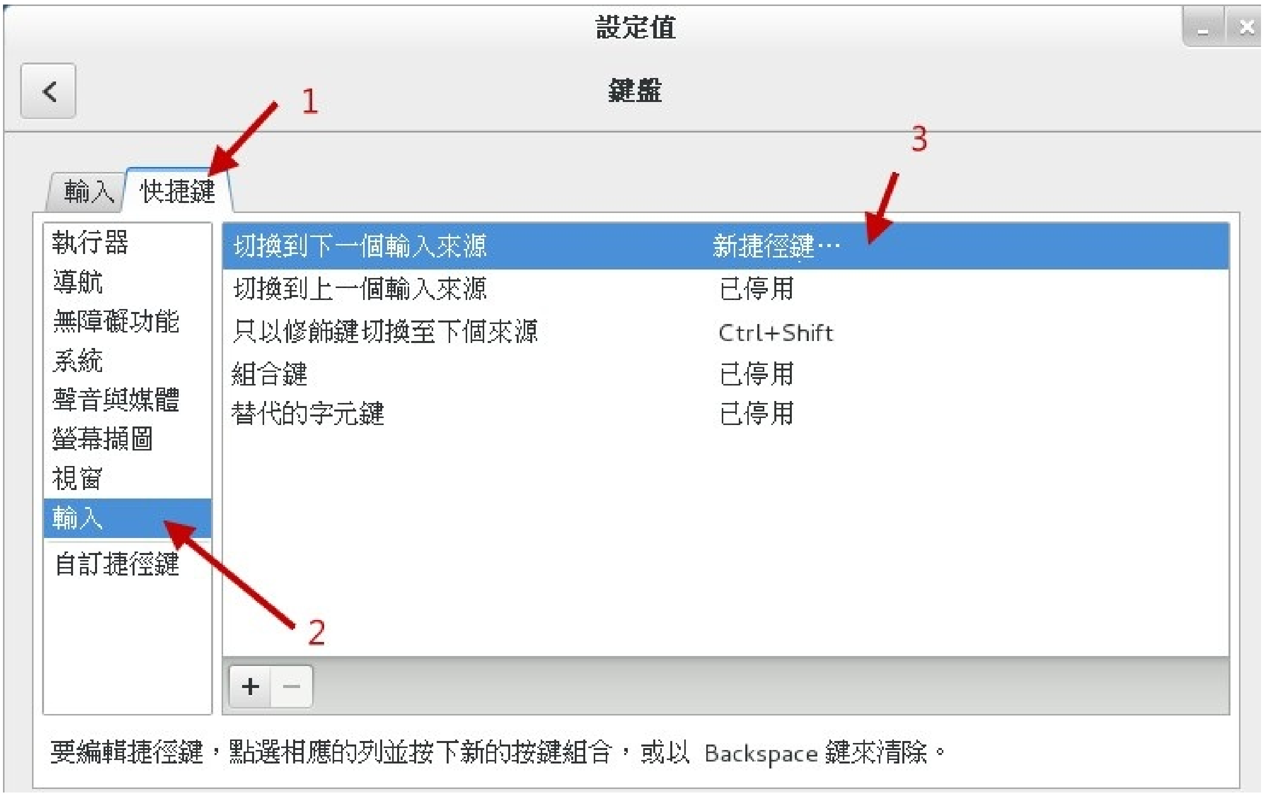


图4.1.17、输入法切换之快捷键设置

- 一些常见的练习

下面的例题请大家自行参考并且实作一下喔！题目很简单，所以鸟哥就不额外抓图了！

1. 由“设置值”的“显示器”项目中，确认一下目前的分辨率，并且尝试自己变更一下屏幕分辨率；
2. 由“设置值”的“背景”项目中，修改一下桌面的背景图示；
3. 由“设置值”的“电源”项目中，修改一下进入空白屏幕锁定的时间，将它改成“永不”的设置值；
4. 由“应用程序”的“公用程序”项目下的“调校工具”中，使用“Shell”功能内的“动态工作区”项目，将原本的 4 个虚拟桌面， 更改成 6 个虚拟桌面看看；
5. 由“应用程序”的“公用程序”项目下的“调校工具”中，使用“输入”项目，并选择“砍除 X 服务器的按键序列”从“已停用”改成“Control+Alt+退格键”的设置， 这可以让你按下三个按钮就能够重新启动 X 窗口管理员；
6. 请将/etc/crontab这个文件“复制”到你的主文件夹中；
7. 从“应用程序”的“附属应用”点选“gedit”编辑器，按下 gedit 的“打开”按钮，选择“主文件夹（就是你的帐号名称）”后，点选刚刚复制过来的 crontab 文件名。 在画面中随意使用中文输入法输入几个字，然后储存离开看看！
8. 从“应用程序”的“喜好”当中打开“终端机”，在终端机中输入“gsettings set org.gnome.desktop.interface enable-animations false”， 这个动作会将 GNOME 默认的画面切换的动画功能关闭，在虚拟机的环境下，有助于画面切换的速度喔！

上述的练习中，第三个练习还挺重要的！因为在默认的状态中，你的图形接口会在 5 分钟后自动的被锁定！这是为了要避免你暂时离开座位， 有人偷偷使用你的计算机的缘故。而要解开锁定，就得要输入你这个帐号的密码才行。这个功能最好是不要取消。但因为我们的系统是单纯的练习机， 而且又是虚拟机，如果经常锁定屏幕，老是要解开很烦～那就使用上述的 3 号练习题，应该可以处理完毕！ 至于第 8 点对于初次接触 Linux 的朋友来说，会有点困难，如果你不知道如何下达指令，没关系～等到本章后面的小节读完，你就知道如何处理了！

- 登出 GNOME、重新启动 X 窗口管理员或关机

如果你没有想要继续玩X Window了，那就登出吧！如果不想要继续操作系统了，那就关机吧！如何登出/关机呢？如下图所示，点选右上角你的帐号名称， 然后在出现的画面中去选择即可。要记得的是，登出前最好将所有不需要的程序都关闭了再登出或关机啊！



图4.1.18、离开窗口接口或 Linux 的方式：有登出、锁定与关机

不论是登出还是关闭电源（关机），都会有一个警告窗口来告知你 60 秒内没有任何动作的话，就会被登出了！如下图所示。当然， 你也可以按下确定来进行动作。登出后，系统画面又会回到原本的等待登陆的画面中了！



图4.1.19、离开窗口接口或 Linux 的方式：登出提醒

请注意喔，登出并不是关机！只是让你的帐号离开系统而已喔！

- 重新启动X Window的快速按钮

一般来说，我们是可以手动来直接修改X Window的配置文件的，不过，修改完成之后的设置项目并不会立刻被载入， 必须要重新启动X才行（特别注意，不是重新开机，而是重新启动X！）。那么如何重新启动X呢？ 最简单的方法就是：

- 直接登出，然后再重新登陆即可；
- 在X的画面中直接按下**[Alt] + [Ctrl] + [Backspace]**

第二个方法比较有趣，[backspace]是倒退键，你按下三个按钮后X Window立刻会被重新启动。 如果你的X Window因为不明原因导致有点问题时，也可以利用这个方法来重新启动X喔！不过， 这个方法要生效，必须要先进行本节稍早之前的练习第五题才行哟！

4.1.3 X window与文字模式的切换

我们前面一直谈到的是X Window的窗口管理员环境，那么在这里面有没有纯命令行的环境啊？因为听说服务器通常是纯命令行的啊！ 当然有啊！但是，要怎么切换X Window与文字模式呢？注意喔，通常我们也称文字模式为终端机接口, terminal 或 console喔！Linux默认的情况下会提供六个Terminal来让使用者登陆， 切换的方式为使用：**[Ctrl] + [Alt] + [F1]~[F6]**的组合按钮。

那这六个终端接口如何命名呢，系统会将[F1] ~ [F6]命名为tty1 ~ tty6的操作接口环境。 也就是说，当你按下[crtl] + [Alt] + [F1]这三个组合按钮时 （按着[ctrl]与[Alt]不放，再按下[F1]功能键）， 就会进入到tty1的terminal接口中了。同样的[F2]就是tty2啰！那么如何回到刚刚的X窗口接口呢？很简单啊！按下[Ctrl] + [Alt] + [F1]就可以了！我们整理一下登陆的环境如下：

- [Ctrl] + [Alt] + [F2] ~ [F6] ：命令行登陆 tty2 ~ tty6 终端机；
- [Ctrl] + [Alt] + [F1] ：图形接口桌面。

由于系统默认的登陆界面不同，因此你想要进入 X 的终端机名称也可能会有些许差异。以 CentOS 7 为例，由于我们这次安装的练习机， 默认是启动图形界面的，因此这个 X 窗口将会出现在 tty1 界面中。如果你的 Linux 默认使用纯命令行，那么 tty1~tty6 就会被文字界面占用。



Tips 在 CentOS 7 环境下，当开机完成之后，默认系统只会提供给你一个 tty 而已，因此无论是文字界面还是图形界面，都是会出现在 tty1 喔！ tty2~tty6 其实一开始是不存在的！但是当你要切换时 （按下 [ctrl]+[alt]+[F2]），系统才产生出额外的 tty2, tty3...

若你在纯文本环境中启动 X 窗口，那么图形界面就会出现在当时的那个 tty 上面。举例来说，你在 tty3 登陆系统，然后输入 startx 启动个人的图形界面， 那么这个图形界面就会产生在 tty3 上面！这样说可以理解吗？

```
# 纯命令行下 （不能有 X 存在） 启动窗口界面的作法
[dmtsai@study ~]$ startx
```

不过startx这个指令并非万灵丹，你要让startx生效至少需要下面这几件事情的配合：

- 并没有其他的 X window 被启用；
- 你必须要已经安装了X Window system，并且X server是能够顺利启动的；
- 你最好要有窗口管理员，例如GNOME/KDE或者是阳春的TWM等；

其实，所谓的窗口环境，就是："文字界面加上 X 窗口软件"的组合！因此，文字界面是一定会存在的，只是窗口界面软件就看你要不要启动而已。 所以，我们才有办法在纯文本环境下启动一个个人化的 X 窗口啊！因为这个 startx 是任何人都可以执行的喔！并不一定需要管理员身份的。 所以，是否默认要使用图形界面，只要在后续管理服务的程序中，将" graphical.target "这个目标服务设置为默认，就能够默认使用图形界面啰！



Tips 从这一版 CentOS 7 开始，已经取消了使用多年的 SystemV 的服务管理方式，也就是说，从这一版开始，已经没有所谓的"执行等级 （run level） "的概念了！ 新的管理方法使用的是 systemd 的模式，这个模式将很多的服务进行相依性管理。以文字与图形界面为例，就是要不要加入图形软件的服务启动而已～ 对于熟悉之前 CentOS 6.x 版本的老家伙们，要重新摸一摸 systemd 这个方式喔！因为不再有 /etc/inittab 啰！注意注意！

4.1.4 在终端接口登陆linux

刚刚你如果有按下[Ctrl] + [Alt] + [F2]就可以来到tty2的登陆画面，而如果你并没有启用图形窗口界面的话， 那么默认就是会来到tty1这个环境中。这个纯文本环境的登陆的画面 （鸟哥用 dmtsai 帐号当入） 有点像这样：

```
CentOS Linux 7 （Core）
Kernel 3.10.0-229.el7.x86_64 on an x86_64

study login: dmtsai
Password: &lt;==这里输入你的密码
Last login: Fri May 29 11:55:05 on tty1 &lt;==上次登陆的情况
[dmtsai@study ~]$ _ &lt;==光标闪烁，等待你的指令输入
```

上面显示的内容是这样的：

1. CentOS Linux 7 （Core）： 显示Linux distribution的名称（CentOS）与版本（7）；
2. Kernel 3.10.0-229.el7.x86_64 on an x86_64： 显示 Linux 核心的版本为3.10.0-229.el7.x86_64， 且目前这部主机的硬件等级为x86_64。
3. study login:： 那个study是你的主机名称。我们在第三章安装时有填写主机名称为：study.centos.vbird，主机名称的显示通常只取第一个小数点前的字母，所以就成为study啦！至于login:则是一支可以让我们登陆的程序。 你可以在login:后面输入你的帐号。以鸟哥为例，我输入的就是第三章创建的dmtsai那个帐号啦！ 当然啰，你也可以使用root这个帐号来登陆的。不过“root”这个帐号代表在Linux系统下无穷的权力， 所以尽量不要使用root帐号来登陆啦！
4. Password:： 这一行则在第三行的dmtai输入后才会出现，要你输入密码啰！ 请注意，在输入密码的时候，屏幕上面“不会显示任何的字样！”， 所以不要以为你的键盘坏掉去！ 很多初学者一开始到这里都会拼命的问！啊我的键盘怎么不能用...
5. Last login: Fri May 29 11:55:05 on tty1： 当使用者登陆系统后，系统会列出上一次这个帐号登陆系统的时间与终端机名称！ 建议大家还是得要看看这个信息，是否真的是自己的登陆所致喔！
6. [dmtsai@study ~]$ _： 这一行则是正确登陆之后才显示的讯息， 最左边的 dmtsai 显示的是“目前使用者的帐号”，而@之后接的 study 则是“主机名称”，至于最右边的~则指的是“目前所在的目录”，那个$则是我们常常讲的“提示字符”啦！



Tips 那个 ~ 符号代表的是“使用者的主文件夹”的意思，他是个“变量！” 这相关的意义我们会在后续的章节依序介绍到。举例来说，root的主文件夹在/root， 所以 ~ 就代表/root的意思。而dmtsai的主文件夹在/home/dmtsai， 所以如果你以dmtsai登陆时，他看到的 ~ 就会等于/home/dmtsai喔！

至于提示字符方面，在Linux当中，默认root的提示字符为 # ，而一般身份使用者的提示字符为 $ 。

还有，上面的第一、第二行的内容其实是来自于/etc/issue这个文件喔！

好了这样就是登陆主机了！很快乐吧！耶～

另外，再次强调，在Linux系统下最好常使用一般帐号来登陆即可，所以上例中鸟哥是以自己的帐号dmtsai来登陆的。 因为系统管理员帐号（root）具有无穷大的权力，例如他可以删除任何一个文件或目录。因此若你以root身份登陆Linux系统， 一个不小心下错指令，这个时候可不是“欲哭无泪”就能够解决的了问题的～

因此，一个称职的网络/系统管理人员，通常都会具有两个帐号，平时以自己的一般帐号来使用Linux主机的任何资源， 有需要动用到系统功能修订时，才会转换身份成为root呢！所以，鸟哥强烈建议你创建一个普通的帐号来供自己平时使用喔！ 更详细的帐号讯息，我们会在后续的“第十三章帐号管理”再次提及！ 这里先有概念即可！

那么如何离开系统呢？其实应该说“登出Linux”才对！登出很简单，直接这样做：

```
[dmtsai@study ~]$ exit
```

就能够登出Linux了。但是请注意：“离开系统并不是关机！” 基本上，Linux本身已经有相当多的工作在进行，你的登陆也仅是其中的一个“工作”而已， 所以当你离开时，这次这个登陆的工作就停止了，但此时Linux其他的工作是还是继续在进行的！ 本章后面我们再来提如何正确的关机，这里先创建起这个概念即可！

4.2 文字模式下指令的下达

其实我们都是通过“程序”在跟系统作沟通的，本章上面提到的窗口管理员或文字模式都是一组或一只程序在负责我们所想要完成的任务。 文字模式登陆后所取得的程序被称为壳（Shell），这是因为这支程序负责最外面跟使用者（我们）沟通，所以才被戏称为壳程序！更多与操作系统及壳程序的相关性可以参考第零章、计算机概论内的说明。

我们Linux的壳程序就是厉害的bash这一支！关于更多的bash我们在第三篇再来介绍。现在让我们来练一练打字吧！



Tips “练打字”真的是开玩笑的！各位观众朋友，千万不要只是“观众朋友”而已，您得要自己亲身体验，看看指令下达之后所输出的信息， 并且理解一下“我敲这个指令的目的是想要完成什么任务？”，再看看输出的结果是否符合你的需求，这样才能学到东西！不是单纯的鸟哥写什么， 你就打什么，那只是“练打字”不是“学Linux”喔！ ^_^

4.2.1 开始下达指令

其实整个指令下达的方式很简单，你只要记得几个重要的概念就可以了。举例来说，你可以这样下达指令的：

```
[dmtsai@study ~]$ command  [-options]  parameter1  parameter2 ...
                   指令        选项          参数（1）      参数（2）
```

上述指令详细说明如下：

1. 一行指令中第一个输入的部分绝对是“指令（command）”或“可可执行文件案（例如批次脚本,script）”
2. command 为指令的名称，例如变换工作目录的指令为 cd 等等；
3. 中刮号[]并不存在于实际的指令中，而加入选项设置时，通常选项前会带 - 号，例如 -h；有时候会使用选项的完整全名，则选项前带有 -- 符号，例如 --help；
4. parameter1 parameter2.. 为依附在选项后面的参数，或者是 command 的参数；
5. 指令, 选项, 参数等这几个咚咚中间以空格来区分，不论空几格 shell 都视为一格。所以空格是很重要的特殊字符！；
6. 按下[Enter]按键后，该指令就立即执行。[Enter]按键代表着一行指令的开始启动。
7. 指令太长的时候，可以使用反斜线 （\） 来跳脱[Enter]符号，使指令连续到下一行。注意！反斜线后就立刻接特殊字符，才能跳脱！

8. 其他：
    i. 在 Linux 系统中，英文大小写字母是不一样的。举例来说， cd 与 CD 并不同。
    ii. 更多的介绍等到第十章 bash 时，再来详述。

注意到上面的说明当中，“第一个被输入的数据绝对是指令或者是可执行的文件”！ 这个是很重要的概念喔！还有，按下[Enter]键表示要开始执行此一命令的意思。我们来实际操作一下： 以ls这个“指令”列出“自己主文件夹（~）”下的“所有隐藏文件与相关的文件属性”， 要达成上述的要求需要加入 -al 这样的选项，所以：

```
[dmtsai@study ~]$ ls -al ~
[dmtsai@study ~]$ ls          -al   ~
[dmtsai@study ~]$ ls -a  -l ~
```

上面这三个指令的下达方式是一模一样的执行结果喔！为什么？请参考上面的说明吧！ 关于更详细的文字模式使用方式，我们会在第十章认识BASH再来强调喔！ 此外，请特别留意，在Linux的环境中，“大小写字母是不一样的东西！”也就是说，在Linux下面， VBird与vbird这两个文件是“完全不一样的”文件呢！所以， 你在下达指令的时候千万要注意到指令是大写还是小写。例如当输入下面这个指令的时候，看看有什么现象：

```
[dmtsai@study ~]$ date  &lt;==结果显示日期与时间
[dmtsai@study ~]$ Date  &lt;==结果显示找不到指令
[dmtsai@study ~]$ DATE  &lt;==结果显示找不到指令
```

很好玩吧！只是改变小写成为大写而已，该指令就变的不存在了！ 因此，请千万记得这个状态哟！

- 语系的支持

另外，很多时候你会发现，咦！怎么我输入指令之后显示的结果的是乱码？ 这跟鸟哥说的不一样啊！呵呵！不要紧张～我们前面提到过，Linux是可以支持多国语系的，若可能的话， 屏幕的讯息是会以该支持语系来输出的。但是，我们的终端机接口（terminal）在默认的情况下， 无法支持以中文编码输出数据的。这个时候，我们就得将支持语系改为英文，才能够以英文显示出正确的讯息。 那怎么做呢？你可以这样做：

```
1\. 显示目前所支持的语系
[dmtsai@study ~]$ locale
LANG=zh_TW.utf8              # 语言语系的输出
LC_CTYPE="zh_TW.utf8"        # 下面为许多信息的输出使用的特别语系
LC_NUMERIC=zh_TW.UTF-8
LC_TIME=zh_TW.UTF-8          # 时间方面的语系数据
LC_COLLATE="zh_TW.utf8"
....中间省略....
LC_ALL=                      # 全部的数据同步更新的设置值
# 上面的意思是说，目前的语系（LANG）为zh_TW.UTF-8，亦即台湾繁体中文的万国码
[dmtsai@study ~]$ date
鍪? 5??29 14:24:36 CST 2015  # 纯命令行下，无法显示中文字，所以前面是乱码

2\. 修改语系成为英文语系
[dmtsai@study ~]$ LANG=en_US.utf8
[dmtsai@study ~]$ export LC_ALL=en_US.utf8
# LANG 只与输出讯息有关，若需要更改其他不同的信息，要同步更新 LC_ALL 才行！

[dmtsai@study ~]$ date
Fri May 29 14:26:45 CST 2015 # 顺利显示出正确的英文日期时间啊！

[dmtsai@study ~]$ locale
LANG=en_US.utf8
LC_CTYPE="en_US.utf8"
LC_NUMERIC="en_US.utf8"
....中间省略....
LC_ALL=en_US.utf8
# 再次确认一下，结果出现，确实是en_US.utf8这个英文语系！
```

注意一下，那个“LANG=en*US.utf8*”是连续输入的，等号两边并没有空白字符喔！ 这样一来，就能够在“这次的登陆”察看英文讯息啰！为什么说是“这次的登陆”呢？ 因为，如果你登出*Linux*后，刚刚下达的指令就没有用啦！ ^^，这个我们会在第十章再好好聊一聊的！好啰，下面我们来练习一下一些简单的指令， 好让你可以了解指令下达方式的模式：

4.2.2 基础指令的操作

下面我们立刻来操作几个简单的指令看看啰！同时请注意，我们已经使用了英文语系作为默认输出的语言喔！

- 显示日期与时间的指令： date
- 显示日历的指令： cal
- 简单好用的计算机： bc

- 1. 显示日期的指令： date

如果在命令行中想要知道目前Linux系统的时间，那么就直接在命令行界面输入date即可显示：

```
[dmtsai@study ~]$ date
Fri May 29 14:32:01 CST 2015
```

上面显示的是：星期五, 五月二十九日, 14:32 分, 01秒，在 2015 年的 CST 时区！台湾在CST时区中啦！ 请赶快动手做做看呦！好了，那么如果我想要让这个程序显示出“2015/05/29”这样的日期显示方式呢？ 那么就使用date的格式化输出功能吧！

```
[dmtsai@study ~]$ date +%Y/%m/%d
2015/05/29
[dmtsai@study ~]$ date +%H:%M
14:33
```

那个“+%Y%m%d”就是date指令的一些参数功能啦！很好玩吧！那你问我， 鸟哥怎么知道这些参数的啊？要背起来吗？当然不必啦！下面再告诉你怎么查这些参数啰！

从上面的例子当中我们也可以知道，指令之后的选项除了前面带有减号“-”之外，某些特殊情况下， 选项或参数前面也会带有正号“+”的情况！这部份可不要轻易的忘记了呢！

- 2. 显示日历的指令： cal

那如果我想要列出目前这个月份的月历呢？呵呵！直接给他下达cal即可！

```
[dmtsai@study ~]$ cal
      May 2015
Su Mo Tu We Th Fr Sa
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

除了本月的日历之外，连同今日所在日期处都会有反白的显示呢！真有趣！cal （calendar）这个指令可以做的事情还很多，例如你可以显示整年的月历情况：

```
[dmtsai@study ~]$ cal 2015
                               2015

       January               February                 March
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
             1  2  3    1  2  3  4  5  6  7    1  2  3  4  5  6  7
 4  5  6  7  8  9 10    8  9 10 11 12 13 14    8  9 10 11 12 13 14
11 12 13 14 15 16 17   15 16 17 18 19 20 21   15 16 17 18 19 20 21
18 19 20 21 22 23 24   22 23 24 25 26 27 28   22 23 24 25 26 27 28
25 26 27 28 29 30 31                          29 30 31

        April                   May                   June
Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa   Su Mo Tu We Th Fr Sa
          1  2  3  4                   1  2       1  2  3  4  5  6
 5  6  7  8  9 10 11    3  4  5  6  7  8  9    7  8  9 10 11 12 13
12 13 14 15 16 17 18   10 11 12 13 14 15 16   14 15 16 17 18 19 20
19 20 21 22 23 24 25   17 18 19 20 21 22 23   21 22 23 24 25 26 27
26 27 28 29 30         24 25 26 27 28 29 30   28 29 30
                       31
....（以下省略）....
```

基本上cal这个指令可以接的语法为：

```
[dmtsai@study ~]$ cal [month] [year]
```

所以，如果我想要知道2015年10月的月历，可以直接下达：

```
[dmtsai@study ~]$ cal 10 2015
    October 2015
Su Mo Tu We Th Fr Sa
             1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

那请问今年有没有13月啊？来测试一下这个指令的正确性吧！下达下列指令看看：

```
[dmtsai@study ~]$ cal 13 2015
cal: illegal month value: use 1-12
```

cal竟然会告诉我们“错误的月份，请使用1-12”这样的信息呢！所以， 未来你可以很轻易的就以cal来取得日历上面的日期啰！简直就是万年历啦！ ^_^。 另外，由这个cal指令的练习我们也可以知道，某些指令有特殊的参数存在， 若输入错误的参数，则该指令会有错误讯息的提示，通过这个提示我们可以借以了解指令下达错误之处。 这个练习的结果请牢记在心中喔！

- 3. 简单好用的计算机： bc

如果在文字模式当中，突然想要作一些简单的加减乘除，偏偏手边又没有计算机！这个时候要笔算吗？ 不需要啦！我们的Linux有提供一支计算程序，那就是bc喔。你在命令行输入bc后，屏幕会显示出版本信息， 之后就进入到等待指示的阶段。如下所示：

```
[dmtsai@study ~]$ bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
_ &lt;==这个时候，光标会停留在这里等待你的输入
```

事实上，我们是“进入到bc这个软件的工作环境当中”了！ 就好像我们在Windows里面使用“小算盘”一样！所以，我们下面尝试输入的数据， 都是在bc程序当中在进行运算的动作。所以啰，你输入的数据当然就得要符合bc的要求才行！ 在基本的bc计算机操作之前，先告知几个使用的运算子好了：

-
    - 加法
-
    - 减法
-
    - 乘法
- / 除法
- ^ 指数
- % 余数

好！让我们来使用bc计算一些咚咚吧！

```
[dmtsai@study ~]$ bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
1+2+3+4  &lt;==只有加法时
10
7-8+3
2
10*52
520
10%3     &lt;==计算“余数”
1
10^2
100
10/100   &lt;==这个最奇怪！不是应该是 0.1 吗？
0
quit     &lt;==离开 bc 这个计算器
```

在上表当中，粗体字表示输入的数据，而在每个粗体字的下面就是输出的结果。 咦！每个计算都还算正确，怎么10/100会变成0呢？这是因为bc默认仅输出整数，如果要输出小数点下位数，那么就必须要执行 scale=number ，那个number就是小数点位数，例如：

```
[dmtsai@study ~]$ bc
bc 1.06.95
Copyright 1991-1994, 1997, 1998, 2000, 2004, 2006 Free Software Foundation, Inc.
This is free software with ABSOLUTELY NO WARRANTY.
For details type `warranty'.
scale=3     &lt;==没错！就是这里！！
1/3
.333
340/2349
.144
quit
```

注意啊！要离开bc回到命令提示字符时，务必要输入“quit”来离开bc的软件环境喔！ 好了！就是这样子啦！简单的很吧！以后你可以轻轻松松的进行加减乘除啦！

从上面的练习我们大概可以知道在命令行界面里面下达指令时，会有两种主要的情况：

- 一种是该指令会直接显示结果然后回到命令提示字符等待下一个指令的输入；
- 一种是进入到该指令的环境，直到结束该指令才回到命令提示字符的环境。

我们以一个简单的图示来说明：

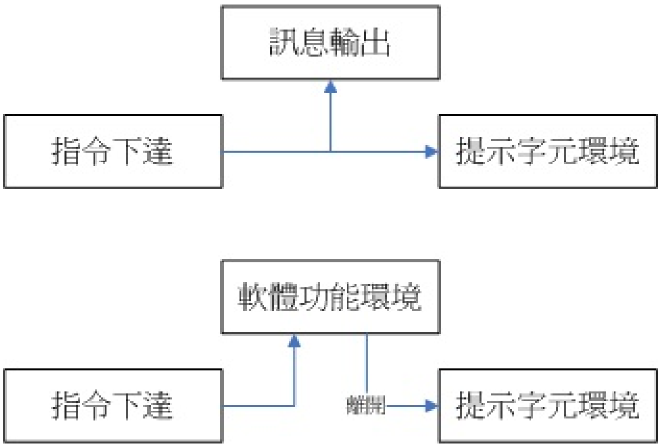


图4.2.1、指令下达的环境，上图为直接显示结果，下图为进入软件功能

如图4.2.1所示，上方指令下达后立即显示讯息且立刻回到命令提示字符的环境。 如果有进入软件功能的环境（例如上面的bc软件），那么就得要使用该软件的结束指令 （例如在bc环境中输入quit）才能够回到命令提示字符中！那你怎么知道你是否在命令提示字符的环境呢？ 很简单！你只要看到光标是在“[dmtsai@study ~]$ ”这种提示字符后面， 那就是等待输入指令的环境了。很容易判断吧！不过初学者还是很容易忘记啦！

4.2.3 重要的几个热键[Tab], [ctrl]-c, [ctrl]-d

在继续后面章节的学习之前，这里很需要跟大家再来报告一件事，那就是我们的文字模式里头具有很多的功能组合键， 这些按键可以辅助我们进行指令的编写与程序的中断呢！这几个按键请大家务必要记住的！很重要喔！

- [Tab]按键

[Tab]按键就是在键盘的大写灯切换按键（[Caps Lock]）上面的那个按键！在各种Unix-Like的Shell当中， 这个[Tab]按键算是Linux的Bash shell最棒的功能之一了！他具有“命令补全”与“文件补齐”的功能喔！ 重点是，可以避免我们打错指令或文件名称呢！很棒吧！但是[Tab]按键在不同的地方输入，会有不一样的结果喔！ 我们举下面的例子来说明。上一小节我们不是提到 cal 这个指令吗？如果我在命令行输入 ca 再按两次 [tab] 按键， 会出现什么讯息？

```
[dmtsai@study ~]$ ca[tab][tab]    &lt;==[tab]按键是紧接在 a 字母后面！
cacertdir_rehash      cairo-sphinx          cancel                case
cache_check           cal                   cancel.cups           cat
cache_dump            calibrate_ppa         capsh                 catchsegv
cache_metadata_size   caller                captoinfo             catman
# 上面的 [tab] 指的是“按下那个tab键”，不是要你输入中括号内的tab啦！
```

发现什么事？所有以ca为开头的指令都被显示出来啦！很不错吧！那如果你输入“ls -al ~/.bash”再加两个[tab]会出现什么？

```
[dmtsai@study ~]$ ls -al ~/.bash[tab][tab]
.bash_history  .bash_logout   .bash_profile  .bashrc
```

咦！在该目录下面所有以 .bash 为开头的文件名称都会被显示出来了呢！注意看上面两个例子喔， 我们按[tab]按键的地方如果是在command（第一个输入的数据）后面时，他就代表着“命令补全”，如果是接在第二个字以后的，就会变成“文件补齐”的功能了！但是在某些特殊的指令下面，文件补齐的功能可能会变成“参数/选项补齐”喔！ 我们同样使用 date 这个指令来查一下：

```
[dmtsai@study ~]$ date --[tab][tab]  &lt;==[tab]按键是紧接在 -- 后面！
--date         --help         --reference=  --rfc-3339=   --universal
--date=        --iso-8601     --rfc-2822    --set=        --version
# 瞧！系统会列出来 date 这个指令可以使用的选项有哪些喔～包括未来会用到的 --date 等项目
```

总结一下：

- [Tab] 接在一串指令的第一个字的后面，则为“命令补全”；
- [Tab] 接在一串指令的第二个字以后时，则为“文件补齐”！
- 若安装 bash-completion 软件，则在某些指令后面使用 [tab] 按键时，可以进行“选项/参数的补齐”功能！

善用 [tab] 按键眞的是个很好的习惯！可以让你避免掉很多输入错误的机会！



Tips 在这一版的 CentOS 7.x 当中，由于多了一个名为 bash_completion 的软件，这个软件会主动的去侦测“各个指令可以下达的选项与参数”等行为， 因此，那个“文件补齐”的功能可能会变成“选项、参数补齐”的功能，不一定会主动补齐文件名了喔！这点得要特别留意。鸟哥第一次接触 CentOS 7 的时候， 曾经为了无法补齐文件名而觉得奇怪！烦恼了老半天说！

- [Ctrl]-c 按键

如果你在Linux下面输入了错误的指令或参数，有的时候这个指令或程序会在系统下面“跑不停”这个时候怎么办？别担心， 如果你想让当前的程序“停掉”的话，可以输入：[Ctrl]与c按键（先按着[Ctrl]不放，且再按下c按键，是组合按键）， 那就是中断目前程序的按键啦！举例来说，如果你输入了“find /”这个指令时，系统会开始跑一些东西（先不要理会这个指令串的意义），此时你给他按下 [Ctrl]-c 组合按键，嘿嘿！是否立刻发现这个指令串被终止了！就是这样的意思啦！

```
[dmtsai@study ~]$ find /
....（一堆东西都省略）....
# 此时屏幕会很花，你看不到命令提示字符的！直接按下[ctrl]-c即可！
[dmtsai@study ~]$ &lt;==此时提示字符就会回来了！find程序就被中断！
```

不过你应该要注意的是，这个组合键是可以将正在运行中的指令中断的， 如果你正在运行比较重要的指令，可别急着使用这个组合按键喔！ ^_^

- [Ctrl]-d 按键

那么[Ctrl]-d是什么呢？就是[Ctrl]与d按键的组合啊！这个组合按键通常代表着：“键盘输入结束（End Of File, EOF 或 End Of Input）”的意思！ 另外，他也可以用来取代exit的输入呢！例如你想要直接离开命令行，可以直接按下[Ctrl]-d就能够直接离开了（相当于输入exit啊！）。

- [shift]+{[PageUP]|[Page Down]}按键

如果你在纯文本的画面中执行某些指令，这个指令的输出讯息相当长啊！所以导致前面的部份已经不在目前的屏幕画面中， 所以你想要回头去瞧一瞧输出的讯息，那怎办？其实，你可以使用 [Shift]+[Page Up] 来往前翻页，也能够使用 [Shift]+[Page Down] 来往后翻页！ 这两个组合键也是可以稍微记忆一下，在你要稍微往前翻画面时，相当有帮助！



Tips 因为目前学生比较常用图形界面的终端机系统，所以当鸟哥谈到 [Shift]+[Page UP] 的功能时，他们很不能理解耶！说都有鼠标滚轮了， 要这组合钮干麻？唉～真是没见过世面的小朋友...

总之，在Linux下面，命令行的功能是很强悍的！要多多的学习他，而要学习他的基础要诀就是...多使用、多熟悉啦！

4.2.4 错误讯息的察看

万一我下达了错误的指令怎么办？不要紧呀！你可以借由屏幕上面显示的错误讯息来了解你的问题点， 那就很容易知道如何改善这个错误讯息啰！举个例子来说，假如想执行date却因为大小写打错成为DATE时， 这个错误的讯息是这样显示的：

```
[dmtsai@study ~]$ DATE
bash: DATE: command not found...   # 这里显示错误的讯息
Similar command is: 'date'         # 这里竟然给你一个可能的解决方案耶！
```

上面那个bash:表示的是我们的Shell的名称， 本小节一开始就谈到过Linux的默认壳程序就是bash啰！ 那么上面的例子说明了bash有错误，什么错误呢？bash告诉你：

> DATE: command not found

字面上的意思是说“指令找不到”，那个指令呢？就是DATE这个指令啦！ 所以说，系统上面可能并没有DATE这个指令啰！就是这么简单！通常出现“command not found”的可能原因为：

- 这个指令不存在，因为该软件没有安装之故。解决方法就是安装该软件；
- 这个指令所在的目录目前的用户并没有将他加入指令搜寻路径中，请参考第十章bash的

PATH说明；

- 很简单！因为你打错字！

从 CentOS 7 开始，bash 竟然会尝试帮我们找解答耶！看一下上面输出的第二行“Similar command is: 'date'”，他说，相似的指令是 date 喔！ 没错啊！我们就是输入错误的大小写而已～这就已经帮我们找到答案了！看了输出，你也应该知道如何解决问题了吧？

介绍这几个指令让你玩一玩先，更详细的指令操作方法我们会在第三篇的时候再进行介绍！现在让我们来想一想，万一我在操作date这个指令的时候，手边又没有这本书，我要怎么知道要如何加那些奇怪的参数， 好让输出的结果符合我想要的输出格式呢？嘿嘿！到下一节鸟哥来告诉你怎么办吧！

4.3 Linux系统的线上求助man page与info page

先来了解一下Linux有多少指令呢？在文字模式下，你可以输入 g 之后直接按下两个[Tab]按键，看看总共有多少以 g 开头的指令可以让你用？



Tips 在这一版中，不输入任何字仅按下两次 [tab] 按钮来显示所有指令的功能被取消了！所以鸟哥以 g 为开头来说明一下啰！

```
[dmtsai@study ~]$ g[tab][tab]&lt;==在g之后直接输入两次[tab]按键
Display all 217 possibilities? （y or n） &lt;==如果不想要看，按 n 离开
```

如上所示，鸟哥安装的这个系统中，少说也有200多个以 g 为开头的指令可以让dmtsai这个帐号使用。 那在Linux里面到底要不要背“指令”啊？可以啊！你背啊！这种事，鸟哥这个“忘性”特佳的老人家实在是背不起来 @_@ ～当然啦，有的时候为了要考试（例如一些认证考试等等的）还是需要背一些重要的指令与选项的！ 不过，鸟哥主要还是以理解“在什么情况下，应该要使用哪方面的指令”为准的！

既然鸟哥说不需要背指令，那么我们如何知道每个指令的详细用法？还有，某些配置文件的内容到底是什么？ 这个可就不需要担心了！因为在Linux上开发的软件大多数都是自由软件/开源软件，而这些软件的开发者为了让大家能够了解指令的用法， 都会自行制作很多的文件，而这些文件也可以直接在线上就能够轻易的被使用者查询出来喔！很不赖吧！ 这根本就是“线上说明文档”嘛！哈哈！没错！确实如此。我们下面就来谈一谈，Linux到底有多少的线上文件数据呢？

4.3.1 指令的 --help 求助说明

事实上，几乎 Linux 上面的指令，在开发的时候，开发者就将可以使用的指令语法与参数写入指令操作过程中了！你只要使用“ --help ”这个选项， 就能够将该指令的用法作一个大致的理解喔！举例来说，我们来瞧瞧 date 这个指令的基本用法与选项参数的介绍：

```
[dmtsai@study ~]# date --help
Usage: date [OPTION]... [+FORMAT]                           # 这里有基本语法
  or:  date [-u&#124;--utc&#124;--universal] [MMDDhhmm[[CC]YY][.ss]]  # 这是设置时间的语法
Display the current time in the given FORMAT, or set the system date.
# 下面是主要的选项说明
Mandatory arguments to long options are mandatory for short options too.
  -d, --date=STRING         display time described by STRING, not 'now'
  -f, --file=DATEFILE       like --date once for each line of DATEFILE
....（中间省略）....
  -u, --utc, --universal    print or set Coordinated Universal Time （UTC）
      --help     显示此求助说明并离开
      --version  显示版本信息并离开
# 下面则是重要的格式 （FORMAT） 的主要项目
FORMAT controls the output.  Interpreted sequences are:

  %%   a literal %
  %a   locale's abbreviated weekday name （e.g., Sun）
  %A   locale's full weekday name （e.g., Sunday）
....（中间省略）....
# 下面是几个重要的范例 （Example）
Examples:
Convert seconds since the epoch （1970-01-01 UTC） to a date
  $ date --date='@2147483647'
....（下面省略）....
```

看一下上面的显示，首先一开始是下达语法的方式 （Usage），这个 date 有两种基本语法，一种是直接下达并且取得日期回传值，且可以 +FORAMAT 的方式来显示。 至于另一种方式，则是加上 MMDDhhmmCCYY 的方式来设置日期时间。他的格式是“月月日日时时分分西元年”的格式！再往下看， 会说明主要的选项，例如 -d 的意义等等，后续又会出现 +FORMAT 的用法！从里面你可以查到我们之前曾经用过得“ date +%Y%m%d ”这个指令与选项的说明。

基本上，如果是指令，那么通过这个简单的 --help 就可以很快速的取得你所需要的选项、参数的说明了！这很重要！我们说过，在 linux 下面你需要学习“任务达成”的方式， 不用硬背指令参数。不过常用的指令你还是得要记忆一下，而选项就通过 --help 来快速查询即可。

同样的，通过 cal --help 你也可以取得相同的解释！相当好用！不过，如果你使用 bc --help 的话，虽然也有简单的解释，但是就没有类似 scale 的用法说明， 同时也不会有 +, -, *, /, % 等运算子的说明了！因此，虽然 --help 已经相当好用，不过，通常 --help 用在协助你查询“你曾经用过的指令所具备的选项与参数”而已， 如果你要使用的是从来没有用过得指令，或者是你要查询的根本就不是指令，而是文件的“格式”时，那就得要通过 man page 啰！！

4.3.2 man page

咦！date --help 没有告诉你 STRING 是什么？嘿嘿！不要担心，除了 --help 之外，我们Linux 上面的其他线上求助系统已经都帮你想好要怎么办了， 所以你只要使用简单的方法去寻找一下说明的内容，马上就清清楚楚的知道该指令的用法了！怎么看呢？就是找男人（man） 呀！喔！不是啦！这个man是manual（操作说明）的简写啦！只要下达：“man date” 马上就会有清楚的说明出现在你面前喔！如下所示：

```
[dmtsai@study ~]$ LANG="en_US.utf8"
```

```
# 还记得这个咚咚的用意吧？前面提过了，是为了“语系”的需要啊！下达过一次即可！

[dmtsai@study ~]$ man date
DATE（1）                         User Commands                        DATE（1）
# 请注意上面这个括号内的数字
NAME  &lt;==这个指令的完整全名，如下所示为date且说明简单用途为设置与显示日期/时间
       date - print or set the system date and time

SYNOPSIS  &lt;==这个指令的基本语法如下所示
       date [OPTION]... [+FORMAT]                           &lt;==第一种单纯显示的用法
       date [-u&#124;--utc&#124;--universal] [MMDDhhmm[[CC]YY][.ss]]   &lt;==这种可以设置系

DESCRIPTION  &lt;==详细说明刚刚语法谈到的选项与参数的用法
       Display the current time in the given FORMAT, or set the system date.

       Mandatory arguments to long options are mandatory for short options too.

       -d, --date=STRING  &lt;==左边-d为短选项名称，右边--date为完整选项名称
              display time described by STRING, not 'now'

       -f, --file=DATEFILE
              like --date once for each line of DATEFILE

       -I[TIMESPEC], --iso-8601[=TIMESPEC]
              output  date/time  in  ISO  8601  format.   TIMESPEC='date' for date only
              default）, 'hours', 'minutes', 'seconds', or 'ns' for  date  and  time  to
              indicated precision.
....（中间省略）....
       # 找到了！下面就是格式化输出的详细数据！
       FORMAT controls the output.  Interpreted sequences are:

       %%     a literal %

       %a     locale's abbreviated weekday name （e.g., Sun）

       %A     locale's full weekday name （e.g., Sunday）
....（中间省略）....
ENVIRONMENT  &lt;==与这个指令相关的环境参数有如下的说明
       TZ     Specifies the timezone, unless overridden by command line parameters.
              If neither is specified, the setting from /etc/localtime is used.

EXAMPLES     &lt;==一堆可用的范本
       Convert seconds since the epoch （1970-01-01 UTC） to a date

              $ date --date='@2147483647'
....（中间省略）....

DATE STRING  &lt;==上面曾提到的 --date 的格式说明！
       The --date=STRING is a mostly free format human readable date string such as "Sun,
       Feb 2004 16:21:42 -0800" or "2004-02-29 16:21:42" or even "next  Thursday".   A  d
       string  may  contain  items  indicating calendar date, time of day, time zone, day

AUTHOR  &lt;==这个指令的作者啦！
       Written by David MacKenzie.

COPYRIGHT  &lt;==受到著作权法的保护！用的就是 GPL 了！
       Copyright © 2013 Free Software Foundation, Inc.  License GPLv3+: GNU GPL version 3
       later &lt;http://gnu.org/licenses/gpl.html&gt;.
       This  is free software: you are free to change and redistribute it.  There is NO W
       RANTY, to the extent permitted by law.

SEE ALSO  &lt;==这个重要，你还可以从哪里查到与date相关的说明文档之意
       The full documentation for date is maintained as a Texinfo manual.  If the  info
       date programs are properly installed at your site, the command

              info coreutils 'date invocation'

       should give you access to the complete manual.

GNU coreutils 8.22                    June 2014                                 DATE
```



Tips 进入man指令的功能后，你可以按下“空白键”往下翻页，可以按下“ q ”按键来离开man的环境。 更多在man指令下的功能，本小节后面会谈到的！

看（鸟哥没骂人！）马上就知道一大堆的用法了！如此一来，不就可以知道date的相关选项与参数了吗？真方便！ 而出现的这个屏幕画面，我们称呼他为man page， 你可以在里头查询他的用法与相关的参数说明。如果仔细一点来看这个man page的话，你会发现几个有趣的东西。

首先，在上个表格的第一行，你可以看到的是：“DATE（1）”，DATE我们知道是指令的名称， 那么（1）代表什么呢？他代表的是“一般使用者可使用的指令”的意思！咦！还有这个用意啊！呵呵！ 没错～在查询数据的后面的数字是有意义的喔！他可以帮助我们了解或者是直接查询相关的数据。 常见的几个数字的意义是这样的：

代号	代表内容
1	使用者在shell环境中可以操作的指令或可可执行文件
2	系统核心可调用的函数与工具等
3	一些常用的函数（function）与函数库（library），大部分为C的函数库（libc）
4	设备文件的说明，通常在/dev下的文件
5	配置文件或者是某些文件的格式
6	游戏（games）
7	惯例与协定等，例如Linux文件系统、网络协定、ASCII code等等的说明
8	系统管理员可用的管理指令
9	跟kernel有关的文件

上述的表格内容可以使用“man man”来更详细的取得说明。通过这张表格的说明， 未来你如果使用man page在察看某些数据时，就会知道该指令/文件所代表的基本意义是什么了。 举例来说，如果你下达了“man null”时，会出现的第一行是：“NULL（4）”，对照一下上面的数字意义， 嘿嘿！原来null这个玩意儿竟然是一个“设备文件”呢！很容易了解了吧！



Tips 上表中的1, 5, 8这三个号码特别重要，也请读者要将这三个数字所代表的意义背下来喔！

再来，man page的内容也分成好几个部分来加以介绍该指令呢！就是上头man date那个表格内， 以NAME作为开始介绍，最后还有个SEE ALSO来作为结束。基本上，man page大致分成下面这几个部分：

代号	内容说明
NAME	简短的指令、数据名称说明
SYNOPSIS	简短的指令下达语法（syntax）简介
DESCRIPTION	较为完整的说明，这部分最好仔细看看！
OPTIONS	针对 SYNOPSIS 部分中，有列举的所有可用的选项说明
COMMANDS	当这个程序（软件）在执行的时候，可以在此程序（软件）中下达的指令
FILES	这个程序或数据所使用或参考或链接到的某些文件
SEE ALSO	可以参考的，跟这个指令或数据有相关的其他说明！
EXAMPLE	一些可以参考的范例

有时候除了这些外，还可能会看到Authors与Copyright等，不过也有很多时候仅有NAME与DESCRIPTION等部分。 通常鸟哥在查询某个数据时是这样来查阅的：

1. 先察看NAME的项目，约略看一下这个数据的意思；
2. 再详看一下DESCRIPTION，这个部分会提到很多相关的数据与使用时机，从这个地方可以学到很多小细节呢；
3. 而如果这个指令其实很熟悉了（例如上面的date），那么鸟哥主要就是查询关于OPTIONS的部分了！ 可以知道每个选项的意义，这样就可以下达比较细部的指令内容呢！
4. 最后，鸟哥会再看一下，跟这个数据有关的还有哪些东西可以使用的？举例来说，上面的SEE ALSO就告知我们还可以利用“info coreutils date”来进一步查阅数据；
5. 某些说明内容还会列举有关的文件（FILES 部分）来提供我们参考！这些都是很有帮助的！

大致上了解了man page的内容后，那么在man page当中我还可以利用哪些按键来帮忙查阅呢？首先， 如果要向下翻页的话，可以按下键盘的空白键，也可以使用[Page Up]与[Page Down]来翻页呢！同时，如果你知道某些关键字的话， 那么可以在任何时候输入“/word”，来主动搜寻关键字！例如在上面的搜寻当中，我输入了“/date”会变成怎样？

```
DATE (1)                          User Commands                          DATE (1)

NAME
       date - print or set the system date and time

SYNOPSIS
       date [OPTION]... [+FORMAT]
       date [-u&#124;--utc&#124;--universal] [MMDDhhmm[[CC]YY][.ss]]

DESCRIPTION
       Display  the  current  time  in  the given FORMAT, or set the system date.

....(中间省略)....

/date &lt;==只要按下/，光标就会跑到这个地方来，你就可以开始输入搜寻字串咯
```

看到了吗，当你按下“/”之后，光标就会移动到屏幕的最下面一行， 并等待你输入搜寻的字串了。此时，输入date后，man page就会开始搜寻跟date有关的字串， 并且移动到该区域呢！很方便吧！最后，如果要离开man page时，直接按下“ q ”就能够离开了。 我们将一些在man page常用的按键给他整理整理：

按键	进行工作
空白键	向下翻一页
[Page Down]	向下翻一页
[Page Up]	向上翻一页
[Home]	去到第一页
[End]	去到最后一页
/string	向“下”搜寻 string 这个字串，如果要搜寻 vbird 的话，就输入 /vbird
?string	向“上”搜寻 string 这个字串
n, N	利用 / 或 ? 来搜寻字串时，可以用 n 来继续下一个搜寻 （不论是 / 或 ?） ，可以利用 N 来进行“反向”搜寻。举例来说，我以 /vbird 搜寻 vbird 字串， 那么可以 n 继续往下查询，用 N 往上查询。若以 ?vbird 向上查询 vbird 字串， 那我可以用 n 继续“向上”查询，用 N 反向查询。
q	结束这次的 man page

要注意喔！上面的按键是在man page的画面当中才能使用的！ 比较有趣的是那个搜寻啦！我们可以往下或者是往上搜寻某个字串，例如要在man page内搜寻vbird这个字串， 可以输入 /vbird 或者是 ?vbird ，只不过一个是往下而一个是往上来搜寻的。而要 重复搜寻 某个字串时，可以使用 n 或者是 N 来动作即可呢！ 很方便吧！^_^

既然有man page，自然就是因为有一些文件数据，所以才能够以man page读出来啰！那么这些man page的数据 放在哪里呢？不同的distribution通常可能有点差异性，不过，通常是放在/usr/share/man这个目录里头，然而，我们可以通过修改他的man page搜寻路径来改善这

个目录的问题！修改/etc/man_db.conf （有的版本为man.conf或manpath.conf或man.config等）即可啰！至于更多的关于man的讯息你可以使用“ man man ”来查询呦！关于更详细的设置，我们会在第十章 bash 当中继续的说明喔！

- 搜寻特定指令/文件的man page说明文档

在某些情况下，你可能知道要使用某些特定的指令或者是修改某些特定的配置文件，但是偏偏忘记了该指令的完整名称。 有些时候则是你只记得该指令的部分关键字。这个时候你要如何查出来你所想要知道的man page呢？ 我们以下面的几个例子来说明man这个指令有用的地方喔！

例题：你可否查出来，系统中还有哪些跟“man”这个指令有关的说明文档呢？答：你可以使用下面的指令来查询一下：

```
[dmtsai@study ~]$ man -f man
man （1）             - an interface to the on-line reference manuals
man （1p）            - display system documentation
man （7）             - macros to format man pages
```

使用 -f 这个选项就可以取得更多与man相关的信息，而上面这个结果当中也有提示了 （数字） 的内容， 举例来说，第三行的“ man （7） ”表示有个man （7）的说明文档存在喔！但是却有个man （1）存在啊！ 那当我们下达“ man man ”的时候，到底是找到哪一个说明文档呢？ 其实，你可以指定不同的文件的，举例来说，上表当中的两个 man 你可以这样将他的文件叫出来：

```
[dmtsai@study ~]$ man 1 man  &lt;==这里是用 man（1） 的文件数据
[dmtsai@study ~]$ man 7 man  &lt;==这里是用 man（7） 的文件数据
```

你可以自行将上面两个指令输入一次看看，就知道，两个指令输出的结果是不同的。 那个1, 7就是分别取出在man page里面关于1与7相关数据的文件文件啰！ 好了，那么万一我真的忘记了下达数字，只有输入“ man man ”时，那么取出的数据到底是1还是7啊？ 这个就跟搜寻的顺序有关了。搜寻的顺序是记录在/etc/man_db.conf 这个配置文件当中， 先搜寻到的那个说明文档，就会先被显示出来！ 一般来说，通常会先找到数字较小的那个啦！因为排序的关系啊！所以， man man 会跟 man 1 man 结果相同！

除此之外，我们还可以利用“关键字”找到更多的说明文档数据喔！什么是关键字呢？ 从上面的“man -f man”输出的结果中，我们知道其实输出的数据是：

- 左边部分：指令（或文件）以及该指令所代表的意义（就是那个数字）；
- 右边部分：这个指令的简易说明，例如上述的“-macros to format man pages”

当使用“man -f 指令”时，man只会找数据中的左边那个指令（或文件）的完整名称，有一点不同都不行！ 但如果我想要找的是“关键字”呢？也就是说，我想要同时找上面说的两个地方的内容，只要该内容有关键字存在， 不需要完全相同的指令（或文件）就能够找到时，该怎么办？请看下个范例啰！

例题：找出系统的说明文档中，只要有man这个关键字就将该说明列出来。答：

```
[dmtsai@study ~]$ man -k man
fallocate (2)          - manipulate file space
zshall (1)             - the Z shell meta-man page
....（中间省略）....
yum-config-manager (1)  - manage yum configuration options and yum repositories
yum-groups-manager (1)  - create and edit yum's group metadata
yum-utils (1)          - tools for manipulating repositories and extended package manageme
```

看到了吧！很多对吧！因为这个是利用关键字将说明文档里面只要含有man那个字眼的（不见得是完整字串） 就将他取出来！很方便吧！ ^_^（上面的结果有特殊字体的显示是为了方便读者查看， 实际的输出结果并不会有特别的颜色显示喔！）

事实上，还有两个指令与man page有关呢！而这两个指令是man的简略写法说～就是这两个：

```
[dmtsai@study ~]$ whatis  [指令或者是数据]   &lt;==相当于 man -f [指令或者是数据]
[dmtsai@study ~]$ apropos [指令或者是数据]   &lt;==相当于 man -k [指令或者是数据]
```

而要注意的是，这两个特殊指令要能使用，必须要有创建 whatis 数据库才行！这个数据库的创建需要以 root 的身份下达如下的指令：

```
[root@study ~]# mandb
# 旧版的 Linux 这个指令是使用 makewhatis 喔！这一版升使用 mandb 了！
```



Tips 一般来说，鸟哥是真的不会去背指令的，只会去记住几个常见的指令而已。那么鸟哥是怎么找到所需要的指令呢？ 举例来说，打印的相关指令，鸟哥其实仅记得 lp （line print）而已。那我就由 man lp 开始，去找相关的说明， 然后，再以 lp[tab][tab] 找到任何以 lp 为开头的指令，找到我认为可能有点相关的指令后，先以 --help 去查基本的用法， 若有需要再以 man 去查询指令的用法！呵呵！所以，如果是实际在管理 Linux ， 那么真的只要记得几个很重要的指令即可，其他需要的，嘿嘿！努力的找男人（man）吧！

4.3.3 info page

在所有的Unix Like系统当中，都可以利用 man 来查询指令或者是相关文件的用法； 但是，在Linux里面则又额外提供了一种线上求助的方法，那就是利用info这个好用的家伙啦！

基本上，info与man的用途其实差不多，都是用来查询指令的用法或者是文件的格式。但是与man page一口气输出一堆信息不同的是，info page则是将文件数据拆成一个一个的段落，每个段落用自己的页面来撰写， 并且在各个页面中还有类似网页的“超链接”来跳到各不同的页面中，每个独立的页面也被称为一个节点（node）。 所以，你可以将info page想成是文字模式的网页显示数据啦！

不过你要查询的目标数据的说明文档必须要以info的格式来写成才能够使用info的特殊功能（例如超链接）。 而这个支持info指令的文件默认是放置在/usr/share/info/这个目录当中的。 举例来说，info这个指令的说明文档有写成info格式，所以，你使用“ info info ”可以得到如下的画面：

```
[dmtsai@study ~]$ info info
File: info.info,  Node: Top,  Next: Getting Started,  Up: （dir）

Info: An Introduction
*********************

The GNU Project distributes most of its on-line manuals in the "Info
format", which you read using an "Info reader".  You are probably using
an Info reader to read this now.
....（中间省略）....

   If you are new to the Info reader and want to learn how to use it,
type the command 'h' now.  It brings you to a programmed instruction
sequence. # 这一段在说明，按下 h 可以有简易的指令说明！很好用！
....（中间省略）....

* Menu:

* Getting Started::             Getting started using an Info reader.
* Advanced::                    Advanced Info commands.
* Expert Info::                 Info commands for experts.
* Index::                       An index of topics, commands, and variables.

--zz-Info: （info.info.gz）Top, 52 lines --Bot---------------------------------------
```

仔细的看到上面这个显示的结果，里面的第一行显示了很多的信息喔！第一行里面的数据意义为：

- File：代表这个info page的数据是来自info.info文件所提供的；
- Node：代表目前的这个页面是属于Top节点。 意思是info.info内含有很多信息，而Top仅是info.info文件内的一个节点内容而已；
- Next：下一个节点的名称为Getting Started，你也可以按“N”到下个节点去；
- Up：回到上一层的节点总揽画面，你也可以按下“U”回到上一层；
- Prev：前一个节点。但由于Top是info.info的第一个节点，所以上面没有前一个节点的信息。

从第一行你可以知道这个节点的内容、来源与相关链接的信息。更有用的信息是，你可以通过直接按下N, P, U来去到下一个、上一个与上一层的节点（node）！非常的方便！ 第一行之后就是针对这个节点的说明。在上表的范例中，第二行以后的说明就是针对info.info内的Top

这个节点所做的。 另外，如论你在任何一个页面，只要不知道怎么使用 info 了，直接按下 h 系统就能够提供一些基本按键功能的介绍喔！

```
     copy of the license to the document, as described in section 6 of
     the license.

* Menu:

* Getting Started::            Getting started using an Info reader.
* Advanced::                   Advanced Info commands.
* Expert Info::                Info commands for experts.
* Index::                      An index of topics, commands, and variables.

--zz-Info: (info.info.gz)Top, 52 lines --Bot--------------------------------------------
Basic Info command keys  # 这里是按下 h 之后才会出现的一堆简易按钮列说明！

x           Close this help window.      # 按下 x 就可以关闭这个 help 的窗口
q           Quit Info altogether.        # 完全离开 info page 喔！
H           Invoke the Info tutorial.

Up          Move up one line.
Down        Move down one line.
DEL         Scroll backward one screenful.
SPC         Scroll forward one screenful.
-----Info: *Info Help*, 405 lines --Top------------------------------------------------
```

再来，你也会看到有“Menu”那个咚咚吧！下面共分为四小节，分别是Getting Started等等的，我们可以使用上下左右按键来将光标移动到该文字或者“ * ”上面，按下Enter， 就可以前往该小节了！另外，也可以按下[Tab]按键，就可以快速的将光标在上表的画面中的node间移动，真的是非常的方便好用。如果将info.info内的各个节点串在一起并绘制成图表的话，情况有点像下面这样：

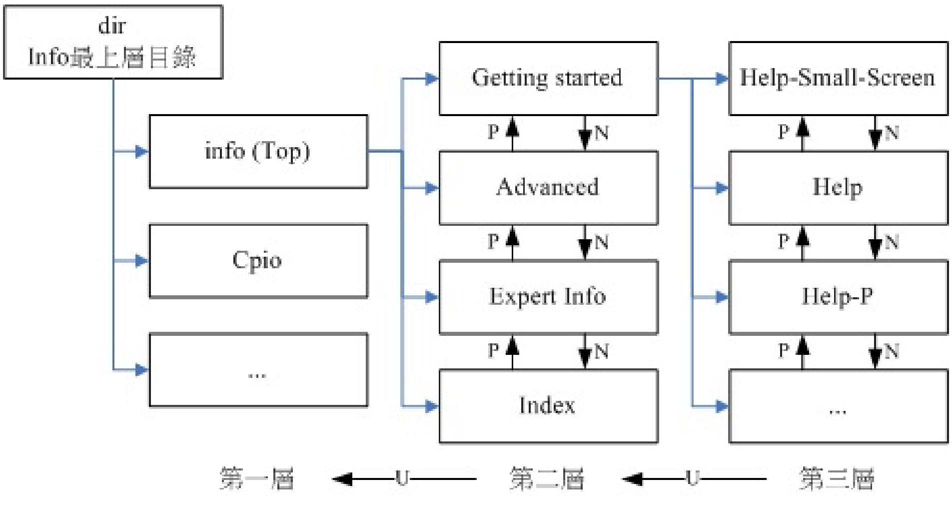


图4.3.1、info page各说明文档相关性的示意图

如同上图所示，info的说明文档将内容分成多个node，并且每个node都有定位与链接。 在各链接之间还可以具有类似“超链接”的快速按钮，可以通过[tab]键在各个超链接间移动。 也可以使用U,P,N来在各个阶层与相关链接中显示！非常的不错用啦！ 至于在info page当中可以使用的按键，可以整理成下面这样，事实上，你也可以在 info page 中按下 h 喔！

按键	进行工作
空白键	向下翻一页
[Page Down]	向下翻一页
[Page Up]	向上翻一页
[tab]	在 node 之间移动，有 node 的地方，通常会以 * 显示。
[Enter]	当光标在 node 上面时，按下 Enter 可以进入该 node 。
b	移动光标到该 info 画面当中的第一个 node 处
e	移动光标到该 info 画面当中的最后一个 node 处
n	前往下一个 node 处
p	前往上一个 node 处
u	向上移动一层
s（/）	在 info page 当中进行搜寻
h, ?	显示求助菜单
q	结束这次的 info page

info page是只有Linux上面才有的产物，而且易读性增强很多～不过查询的指令说明要具有info page功能的话，得用info page的格式来写成线上求助文件才行！我们CentOS 7将info page的文件放置到/usr/share/info/目录中！至于非以info page格式写成的说明文档（就是man page），虽然也能够使用info来显示，不过其结果就会跟man相同。 举例来说，你可以下达“info man”就知道结果了！ ^_^

4.3.4 其他有用的文件（documents）

刚刚前面说，一般而言，指令或者软件制作者，都会将自己的指令或者是软件的说明制作成“线上说明文档”！ 但是，毕竟不是每个咚咚都需要做成线上说明文档的，还有相当多的说明需要额外的文件！此时，这个所谓的 How-To（如何做的意思）就很重要啦！还有，某些软件不只告诉你“如何做”，还会有一些相关的原理会说明呢。

那么这些说明文档要摆在哪里呢？哈哈！就是摆在/usr/share/doc这个目录啦！所以说，你只要到这个目录下面， 就会发现好多好多的说明文档啦！还不需要到网络上面找数据呢！厉害吧！^_^ 举例来说，你可能会先想要知道 grub2 这个新版的开机管理软件有什么能使用的指令？那可以到下面的目录瞧瞧：

- /usr/share/doc/grub2-tools-2.02

另外，很多原版软件释出的时候，都会有一些安装须知、预计工作事项、未来工作规划等等的东西，还有包括可安装的程序等， 这些文件也都放置在 /usr/share/doc 当中喔！而且/usr/share/doc这个目录下的数据主要是以套件（packages）为主的， 例如 nano 这个软件

的相关信息在 /usr/share/doc/nano-xxx（那个xxx表示版本的意思！）。

总结上面的三个咚咚（man, info, /usr/share/doc/），请记住喔：

- 在终端机模式中，如果你知道某个指令，但却忘记了相关选项与参数，请先善用 --help 的功能来查询相关信息；
- 当有任何你不知道的指令或文件格式这种玩意儿，但是你想要了解他，请赶快使用man 或者是info来查询！
- 而如果你想要架设一些其他的服务，或想要利用一整组软件来达成某项功能时，请赶快到/usr/share/doc 下面查一查有没有该服务的说明文档喔！
- 另外，再次的强调，因为Linux毕竟是外国人发明的，所以中文文件确实是比较少的！ 但是不要害怕，拿本英文字典在身边吧！随时查阅！不要害怕英文喔！

4.4 超简单文书编辑器： nano

在Linux系统当中有非常多的文书编辑器存在，其中最重要的就是后续章节我们会谈到的 vim 这家伙！ 不过其实还有很多不错用的文书编辑器存在的！在这里我们就介绍一下简单的nano这一支文书编辑器来玩玩先！

nano的使用其实很简单，你可以直接加上文件名就能够打开一个旧文件或新文件！下面我们就来打开一个名为text.txt的文件名来看看：

```
[dmtsai@study ~]$ nano text.txt
# 不管text.txt存不存在都没有关系！存在就打开旧文件，不存在就打开新文件

 GNU nano 2.3.1                       File: text.txt

   &lt;==这个是光标所在处

                                  [ New File ]
^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Te   ^T To Spell
# 上面两行是指令说明列，其中^代表的是[ctrl]的意思
```

如上图所示，你可以看到第一行反白的部分，那仅是在宣告nano的版本与文件名（File: text.txt）而已。 之后你会看到最下面的三行，分别是文件的状态（New File）与两行指令说明列。指令说明列反白的部分就是组合键， 接的则是该组合键的功能。那个指数符号（^）代表的是键盘的[Ctrl]按键啦！下面先来说说比较重要的几个组合按键：

- [ctrl]-G：取得线上说明（help），很有用的！
- [ctrl]-X：离开naon软件，若有修改过文件会提示是否需要储存喔！
- [ctrl]-O：储存盘案，若你有权限的话就能够储存盘案了；
- [ctrl]-R：从其他文件读入数据，可以将某个文件的内容贴在本文件中；
- [ctrl]-W：搜寻字串，这个也是很有帮助的指令喔！
- [ctrl]-C：说明目前光标所在处的行数与列数等信息；
- [ctrl]-_：可以直接输入行号，让光标快速移动到该行；
- [alt]-Y：校正语法功能打开或关闭（按一下开、再按一下关）
- [alt]-M：可以支持鼠标来移动光标的功能

比较常见的功能是这些，如果你想要取得更完整的说明，可以在nano的画面中按下[ctrl]-G或者是[F1]按键， 就能够显示出完整的naon内指令说明了。好了，请你在上述的画面中随便输入许多字， 输入完毕之后就储存后离开，如下所示：

```
 GNU nano 2.3.1                        File: text.txt

Type some words in this nano editor program.
You can use [ctrl] plus some keywords to go to some functions.
Hello every one.
Bye bye.
   &lt;==这个是由标所在处

^G Get Help   ^O WriteOut   ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos
^X Exit       ^J Justify    ^W Where Is   ^V Next Page  ^U UnCut Te   ^T To Spell
```

此时按下[crtl]-X会出现类似下面的画面：

```
 GNU nano 2.3.1                        File: text.txt

Type some words in this nano editor program.
You can use [ctrl] plus some keywords to go to some functions.
Hello every one.
Bye bye.

Save modified buffer （ANSWERING "No" WILL DESTROY CHANGES） ? █
 Y Yes
 N No           ^C Cancel
```

如果不要储存数据只想要离开，可以按下N即可离开。如果确实是需要储存的，那么按下Y后，最后三行会出现如下画面：

```
File Name to Write: text.txt█  &lt;==可在这里修改文件名或直接按[enter]
^G Get Help           M-D DOS Format       M-A Append           M-B Backup File
^C Cancel             M-M Mac Format       M-P Prepend
```

如果是单纯的想要储存而已，直接按下[enter]即可储存后离开nano程序。不过上表中最下面还有两行，我们知道指数符号代表[crtl]， 那个M是代表什么呢？其实就是[alt]啰！其实nano也不需要记太多指令啦！只要知道怎么进入nano、怎么离开，怎么搜寻字串即可。 未来我们还会学习更有趣的vi呢！

4.5 正确的关机方法

OK！大概知道开机的方法，也知道基本的指令操作，而且还已经知道线上查询了，好累呦！想去休息呢！那么如何关机呢？我想，很多朋友在DOS的年代已经有在玩计算机了！ 在当时我们关掉DOS的系统时，常常是直接关掉电源开关，而 Windows 在你不爽的时候，按着电源开关四秒也可以关机！但是在Linux则相当的不建议这么做！

Why？在 Windows （非 NT 主机系统） 系统中，由于是单人假多任务的情况，所以即使你的计算机关机， 对于别人应该不会有影响才对！不过呢，在 Linux 下面，由于每个程序 （或者说是服务） 都是在在背景下执行的，因此，在你看不到的屏幕背后其实可能有相当多人同时在你的主机上面工作， 例如浏览网页啦、传送信件啦以 FTP 传送文件啦等等的，如果你直接按下电源开关来关机时， 则其他人的数据可能就此中断！那可就伤脑筋了！

此外，最大的问题是，若不正常关机，则可能造成文件系统的毁损 （因为来不及将数据回写到文件中，所以有些服务的文件会有问题！）。所以正常情况下，要关机时需要注意下面几件事：

- 观察系统的使用状态： 如果要看目前有谁在线上，可以下达“who”这个指令，而如果要看网络的连线状态，可以下达“ netstat -a ”这个指令，而要看背景执行的程序可以执行“ ps -aux ”这个指令。使用这些指令可以让你稍微了解主机目前的使用状态！当然啰，就可以让你判断是否可以关机了 （这些指令在后面Linux常用指令中会提及喔！）

- 通知线上使用者关机的时刻： 要关机前总得给线上的使用者一些时间来结束他们的工作，所以，这个时候你可以使用 shutdown 的特别指令来达到此一功能。

- 正确的关机指令使用： 例如 shutdown 与 reboot 两个指令！

所以下面我们就来谈一谈几个与关机/重新开机相关的指令啰！

- 将数据同步写入硬盘中的指令： sync
- 惯用的关机指令： shutdown
- 重新开机，关机： reboot, halt, poweroff



Tips 由于Linux系统的关机/重新开机是很重大的系统运行，因此只有root才能够进行例如shutdown, reboot等指令。 不过在某些distributions当中，例如我们这里谈到的CentOS系统，他允许你在本机前的tty1~tty7当中（无论是文字界面或图形界面）， 可以用一般帐号来关机或重新开机！但某些distributions则在你要关机时，他会要你输入root的密码呢！^_^

- 数据同步写入磁盘： sync

在第零章、计算机概论里面我们谈到过数据在计算机中运行的模式， 所有的数据都得要被读入内存后才能够被CPU所处理，但是数据又常常需要由内存写回硬盘当中（例如储存的动作）。 由于硬盘的速度太慢（相对于内存来说），如果常常让数据在内存与硬盘中来回写入/读出，系统的性能就不会太好。

因此在Linux系统中，为了加快数据的读取速度，所以在默认的情况中， 某些已经载入内存中的数据将不会直接被写回硬盘，而是先暂存在内存当中，如此一来， 如果一个数据被你重复的改写，那么由于他尚未被写入硬盘中，因此可以直接由内存当中读取出来， 在速度上一定是快上相当多的！

不过，如此一来也造成些许的困扰，那就是万一你的系统因为某些特殊情况造成不正常关机（例如停电或者是不小心踢到power）时，由于数据尚未被写入硬盘当中，哇！所以就会造成数据的更新不正常啦！ 那要怎么办呢？这个时候就需要sync这个指令来进行数据的写入动作啦！ 直接在命令行下输入sync，那么在内存中尚未被更新的数据，就会被写入硬盘中！所以，这个指令在系统关机或重新开机之前， 很重要喔！最好多执行几次！

虽然目前的 shutdown/reboot/halt 等等指令均已经在关机前进行了 sync 这个工具的调用，不过，多做几次总是比较放心点～呵呵～

```
[dmtsai@study ~]$ su -    # 这个指令在让你的身份变成 root ！下面请输入 root 的密码！
Password:  # 就这里！请输入安装时你所设置的 root 密码！
Last login: Mon Jun  1 16:10:12 CST 2015 on pts/0

[root@study ~]# sync
```



Tips 事实上sync也可以被一般帐号使用喔！只不过一般帐号使用者所更新的硬盘数据就仅有自己的数据，不像root可以更新整个系统中的数据了。

- 惯用的关机指令： shutdown

由于Linux的关机是那么重要的工作，因此除了你是在主机前面以实体终端机 （tty1~tty7） 来登陆系统时， 不论用什么身份都能够关机之外，若你是使用远端管理工具（如通过pietty使用ssh服务来从其他计算机登陆主机）， 那关机就只有root有权力而已喔！

嗯！那么就来关机试试看吧！我们较常使用的是shutdown这个指令，而这个指令会通知系统内的各个程序 （processes），并且将通知系统中的一些服务来关闭。shutdown可以达成如下的工作：

- 可以自由选择关机模式：是要关机或重新开机均可；

- 可以设置关机时间: 可以设置成现在立刻关机, 也可以设置某一个特定的时间才关机。
- 可以自订关机讯息：在关机之前，可以将自己设置的讯息传送给线上 user 。
- 可以仅发出警告讯息：有时有可能你要进行一些测试，而不想让其他的使用者干扰，或者是明白的告诉使用者某段时间要注意一下！这个时候可以使用 shutdown 来吓一吓使用者，但却不是真的要关机啦！

那么shutdown的语法是如何呢？聪明的读者大概已经开始找“男人”了！没错，随时随地的 man 一下，是很不错的举动！好了，简单的语法规则为：

```
[root@study ~]# /sbin/shutdown [-krhc] [时间] [警告讯息]
选项与参数：
-k      :  不要真的关机，只是发送警告讯息出去！
-r      :  在将系统的服务停掉之后就重新开机（常用）
-h      :  将系统的服务停掉后，立即关机。 （常用）
-c      :  取消已经在进行的 shutdown 指令内容。
时间    :  指定系统关机的时间！时间的范例下面会说明。若没有这个项目，则默认 1 分钟后自动进行。
范例：
[root@study ~]# /sbin/shutdown -h 10 'I will shutdown after 10 mins'
Broadcast message from root@study.centos.vbird （Tue 2015-06-02 10:51:34 CST）:

I will shutdown after 10 mins
The system is going down for power-off at Tue 2015-06-02 11:01:34 CST!
```

在执行 shutdown 之后，系统告诉大家，这部机器会在十分钟后关机！并且会将讯息显示在目前登陆者的屏幕前方！ 你可以输入“ shutdown -c ”来取消这次的关机指令。而如果你什么参数都没有加，单纯执行 shutdown 之后， 系统默认会在 1 分钟后进行“关机”的动作喔！我们也提供几个常见的时间参数给你参考！



Tips 与旧版不同的地方在于，以前 shutdown 后面一定得要加时间参数才行，如果没有加上的话，系统会跳到单人维护模式中。 在这一版中，shutdown 会以 1 分钟为限，进行自动关机的任务！真的很不一样喔！所以时间参数可以不用加啰！

```
[root@study ~]# shutdown -h now
立刻关机，其中 now 相当于时间为 0 的状态
[root@study ~]# shutdown -h 20:25
系统在今天的 20:25 分会关机，若在21:25才下达此指令，则隔天才关机
[root@study ~]# shutdown -h +10
系统再过十分钟后自动关机
[root@study ~]# shutdown -r now
系统立刻重新开机
[root@study ~]# shutdown -r +30 'The system will reboot'
再过三十分钟系统会重新开机，并显示后面的讯息给所有在线上的使用者
[root@study ~]# shutdown -k now 'This system will reboot'
仅发出警告信件的参数！系统并不会关机啦！吓唬人！
```

- 重新开机，关机： reboot, halt, poweroff

还有三个指令可以进行重新开机与关机的任务，那就是reboot, halt, poweroff。 其实这三个指令调用的函数库都差不多，所以当你使用“man reboot”时，会同时出现三个指令的用法给你看呢。 其实鸟哥通常都只有记poweroff与reboot这两个指令啦！一般鸟哥在重新开机时，都会下达如下的指令喔：

```
[root@study ~]# sync; sync; sync; reboot
```

既然这些指令都能够关机或重新开机，那他有没有什么差异啊？基本上，在默认的情况下，这几个指令都会完成一样的工作！（全部的动作都是去调用 systemctl 这个重要的管理命令！） 所以，你只要记得其中一个就好了！重点是，你自己习惯即可！

```
[root@study ~]# halt      # 系统停止～屏幕可能会保留系统已经停止的讯息！
[root@study ~]# poweroff  # 系统关机，所以没有提供额外的电力，屏幕空白！
```

更多halt与poweroff的选项功能，请务必使用man去查询一下喔！

- 实际使用管理工具 systemctl 关机

如果你跟鸟哥一样是个老人家，那么一定会知道有个名为 init 的指令，这个指令可以切换不同的执行等级～ 执行等级共有 0~6 七个，其中 0 就是关机、6 就是重新开机等等。不过，这个 init 目前只是一个相容模式而已～ 所以在 CentOS 7 当中，虽然你依旧可以使用“ init 0 ”来关机，但是那已经跟所谓的“执行等级”无关了！

那目前系统中所有服务的管理是使用哪个指令呢？那就是 systemctl 啦！这个指令相当的复杂！我们会在很后面系统管理员部份才讲的到！ 目前你只要学习 systemctl 当中与关机有关的部份即可。要注意，上面谈到的 halt, poweroff, reboot, shutdown 等等，其实都是调用这个 systemctl 指令的喔！ 这个指令跟关机有关的语法如下：

```
[root@study ~]# systemctl [指令]
指令项目包括如下：
halt        进入系统停止的模式，屏幕可能会保留一些讯息，这与你的电源管理模式有关
poweroff    进入系统关机模式，直接关机没有提供电力喔！
reboot      直接重新开机
suspend     进入休眠模式

[root@study ~]# systemctl reboot    # 系统重新开机
[root@study ~]# systemctl poweroff  # 系统关机
```

4.6 重点回顾

- 为了避免瞬间断电造成的Linux系统危害，建议做为服务器的Linux主机应该加上不断电系统来持续提供稳定的电力；
- 养成良好的操作习惯，尽量不要使用 root 直接登陆系统，应使用一般帐号登陆系统，有需要再转换身份
- 可以通过“活动总览”查看系统所有使用的软件及快速启用惯用软件
- 在X的环境下想要“强制”重新启动X的组合按键为：“[alt]+[ctrl]+[backspace]”；
- 默认情况下，Linux提供tty1~tty6的终端机界面；
- 在终端机环境中，可依据提示字符为$或#判断为一般帐号或root帐号；
- 取得终端机支持的语系数据可下达“echo $LANG”或“locale”指令；
- date可显示日期、cal可显示日历、bc可以做为计算机软件；
- 组合按键中，[tab]按键可做为（1）命令补齐或（2）文件名补齐或（3）参数选项补齐，[crtl]-[c]可以中断目前正在运行中的程序；
- Linux 系统上的英文大小写为不同的数据
- 线上说明系统有man及info两个常见的指令；
- man page说明后面的数字中，1代表一般帐号可用指令，8代表系统管理员常用指令，5代表系统配置文件格式；
- info page可将一份说明文档拆成多个节点（node）显示，并具有类似超链接的功能，增加易读性；
- 系统需正确的关机比较不容易损坏，可使用shutdown, poweroff等指令关机。

4.7 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看）

情境仿真题一：我们在纯命令行，例如tty2里面看到的欢迎画面，就是在那个login:之前的画面（CentOS Linux 7 ...）是怎么来的？

- 目标：了解到终端机接口的欢迎讯息是怎么来的？
- 前提：欢迎讯息的内容，是记录到/etc/issue当中的
- 需求：利用man找到该文件当中的变量内容

情境仿真题一的解决步骤：

1. 欢迎画面是在/etc/issue文件中，你可以使用“nano /etc/issue”看看该文件的内容（注意，不要修改这个文件内容，看完就离开），这个文件的内容有点像下面这样：

````
```
\S
Kernel \r on an \m
```
````

1. 与tty3比较之下，发现到核心版本使用的是 \r 而硬件等级则是 \m 来取代，这两者代表的意义为何？ 由于这个文件的文件名是issue，所以我们使用“man issue”来查阅这个文件的格式；
2. 通过上一步的查询我们会知道反斜线（\）后面接的字符是与agetty（8）及mingetty（8）有关，故进行“man agetty”这个指令的查询。
3. 由于反斜线（\）的英文为“escape”因此在上个步骤的man环境中，你可以使用“/escape”来搜寻各反斜线后面所接字符所代表的意义为何。
4. 请自行找出：如果我想要在/etc/issue文件内表示“时间（localtime）”与“tty号码（如tty1, tty2的号码）”的话，应该要找到那个字符来表示（通过反斜线的功能）？（答案为：\t 与 \l）

简答题部分：

- 简单的查询一下，Physical console / Virtual console / Terminal 的说明为何？console 有“控制台”的意思在里面，因此你可以这样看的：
    - 实体控制台：实体的屏幕、键盘、鼠标等界面，让你可以使用该配备来操作系统的

  环境，就称为实体控制台 （Physical console）
  - 虚拟控制台：由系统衍生出的虚拟控制台，你可以通过该虚拟控制台搭配你自己系统的实体配备，来操作远端系统的环境。每个虚拟控制台都是独立运行的。
  - 终端机：你可以用该界面来取得一个可以控制系统的 shell 环境。

  由这些定义来看，一般来说，我们取得可以与系统互动的环境，大致上都称为 terminal 就是了。

- 请问如果我以文字模式登陆Linux主机时，我有几个终端机接口可以使用？如何切换各个不同的终端机接口？共有六个， tty1 ~ tty6 ，切换的方式为 Crtl + Alt + [F1]~[F6]
- 在Linux系统中，/VBird与/vbird是否为相同的文件？两者为不同的文件，因为 Linux 系统中，大小写字母代表意义不一样！
- 我想要知道 date 如何使用，应该如何查询？最简单的方式就是使用 man date 或 info date 来查看，如果该套件有完整说明的话，那么应该也可以在 /usr/share/doc 里面找到说明文档！
- 我想要在今天的 1:30 让系统自己关机，要怎么做？shutdown -h 1:30
- 如果我 Linux 的 X Window 突然发生问题而挂掉，但 Linux 本身还是好好的，那么我可以按下哪三个按键来让 X window 重新启动？[crtl]+[alt]+[backspace]
- 我想要知道 2010 年 5 月 2 日是星期几？该怎么做？最简单的方式直接使用 cal 5 2010 即可找出 2010 年 5 月份的月历。
- 使用 man date 然后找出显示目前的日期与时间的参数，成为类似：2015/10/16-20:03date +%Y/%m/%d-%H:%M
- 若以 X-Window 为默认的登陆方式，那请问如何进入 Virtual console 呢？可以按下 [Ctrl] + [Alt] + [F2] ~ [F6] 进入 Virtual console （ 共六个 ）； 而按下 [Ctrl] + [Alt] + [F1] 可回到 X-Window 的 desktop 中！
- 简单说明在 bash shell 的环境下， [tab] 按键的用途？[Tab] 按键可做为命令补齐或文件补齐的功能，与所接的指令位置有关。 接在一串指令的第一个单字后面，则为命令补齐，否则则为文件补齐！ 目前尚有选项/参数补齐的功能。
- 如何强制中断一个程序的进行？（利用按键，非利用 kill 指令）可以利用 [Ctrl] + c 来中断！
- Linux 提供相当多的线上查询，称为 man page，请问，我如何知道系统上有多少关于 passwd 的说明？又，可以使用其他的程序来取代 man 的这个功能吗？可以利用 man -f passwd 来查询，另外，如果有提供 info 的文件数据时 （在 /usr/share/info/ 目录中） ，则能够利用 info passwd 来查询之！
- 在 man 的时候， man page 显示的内容中，指令（或文件）后面会接一组数字，这个数字若为 1, 5, 8 ，表示该查询的指令（或文件）意义为何？代表意义为 1） 一般使用者可以使用的指令或可可执行文件案 5）一些配置文件的文件内容格式 8）系统管理员能够使用的管理指令。
- man page 显示的内容的文件是放置在哪些目录中？放置在 /usr/share/man/ 与 /usr/local/man 等默认目录中。
- 请问这一串指令“ foo1 -foo2 foo3 foo4 ”中，各代表什么意义？foo1 一定是指令， -foo2 则是 foo1 这个指令的选择项目参数， foo3 与 foo4 则不一定， 可能是 foo1 的参数设置

值，也可能是额外加入的 parameters。

- 当我输入 man date 时，在我的终端机却出现一些乱码，请问可能的原因为何？如何修正？如果没有其他错误的发生，那么发生乱码可能是因为语系的问题所致。 可以利用 export LANG=en_US.utf8 或者是 export LC_ALL=en_US.utf8 等设置来修订这个问题。
- 我输入这个指令“ls -al /vbird”，系统回复我这个结果：“ls: /vbird: No such file or directory” 请问发生了什么事？”不要紧张，很简单的英文，因为系统根本没有 /vbird 这个文件的存在啊！ ^_^
- 我想知道目前系统有多少指令是以 bz 为开头的，可以怎么作？直接输入 bz[tab][tab] 就可以知道了！
- 承上题，在出现的许多指令中，请问 bzip2 是干嘛用的？在使用 man bzip2 之后，可以发现到，其实 bzip2 是用来作为压缩与解压缩文件用的！
- 在终端机里面登陆后，看到的提示字符 $ 与 # 有何不同？平时操作应该使用哪一个？# 代表以 root 的身份登陆系统，而 $ 则代表一般身份使用者。依据提示字符的不同， 我们可以约略判断登陆者身份。一般来说，建议日常操作使用一般身份使用者登陆，亦即是 $ ！
- 我使用dmtsai这个帐号登陆系统了，请问我能不能使用reboot来重新开机？ 若不能，请说明原因，若可以，请说明指令如何下达？理论上reboot仅能让root执行。不过，如果dmtsai是在主机前面以图形接口登陆时，则dmtsai还是可以通过图形接口功能来关机。

4.8 参考资料与延伸阅读

- 为了让Linux的窗口显示效果更佳，很多团体开始发展桌面应用的环境，GNOME/KDE都是。 他们的目标就是发展出类似Windows桌面的一整套可以工作的桌面环境，他可以进行窗口的定位、放大、缩小、 同时还提供很多的桌面应用软件。下面是KDE与GNOME的相关链接： http://www.kde.org/ http://www.gnome.org/

2002/07/16：第一次完成吧？ 2003/02/06：重新编排与加入 FAQ 2004/05/01：在shutdown 的指令部分，修改 shutdown -k "messages" 成为 shutdown -k now "messages"，很抱歉，写错了！ 2005/06/17：将原本的文章移动到 这里 2005/06/27：终于写完了！写的眞久～没办法，将 man page 扩大解释，增加的幅度还挺多的！ 2005/08/23：刚刚才发现，那个man page的内部指令说明中， n 与 N 的说明错误了！已订正！ 2007/12/08：通过网友sheaushyong的发现，之前将Live CD中，说明要挂载 / 才 fsck 是不对的！ 2008/09/03：将原本的Fedora Core IV的文章移动到此处。 2008/09/08：加入了一些图示说明，尤其是info的部分多了一个示意图！ 2008/09/09：加入了nano这个简单的文书编辑器说明，以及情境仿眞题的解释！ 2009/09/17：修订了显示的信息，将图片重新抓图汇整。 2015/05/21：基于CentOS 5.x 的首次开机说明文档移到这里了！

第五章、Linux 的文件权限与目录配置

最近更新日期：20//

Linux最优秀的地方之一就在于他的多用户多任务环境。而为了让各个使用者具有较保密的文件数据，因此文件的权限管理就变的很重要了。 Linux一般将文件可存取的身份分为三个类别，分别是 owner/group/others，且三种身份各有 read/write/execute 等权限。若管理不当，你的Linux主机将会变的很“不苏湖！@_@”。另外，你如果首次接触Linux的话，那么， 在Linux下面这么多的目录/文件，到底每个目录/文件代表什么意义呢？下面我们就来一一介绍呢！

5.1 使用者与群组

经过第四章的洗礼之后，你应该可以在Linux的命令行界面下面输入指令了吧？ 接下来，当然是要让你好好的浏览一下Linux系统里面有哪些重要的文件啰。 不过，每个文件都有相当多的属性与权限，其中最重要的可能就是文件的拥有者的概念了。 所以，在开始文件相关信息的介绍前，鸟哥先就简单的（1）使用者及（2）群组与（3）非本群组外的其他人等概念作个说明吧～ 好让你快点进入状况的哩！ ^_^

1. 文件拥有者

   初次接触Linux的朋友大概会觉得很怪异，怎么“Linux有这么多使用者， 还分什么群组，有什么用？”。这个“使用者与群组”的功能可是相当健全而好用的一个安全防护呢！怎么说呢？ 由于Linux是个多用户多任务的系统，因此可能常常会有多人同时使用这部主机来进行工作的情况发生， 为了考虑每个人的隐私权以及每个人喜好的工作环境，因此，这个“文件拥有者”的角色就显的相当的重要了！

   例如当你将你的e-mail情书转存成文件之后，放在你自己的主文件夹，你总不希望被其他人看见自己的情书吧？ 这个时候，你就把该文件设置成“只有文件拥有者，就是我，才能看与修改这个文件的内容”， 那么即使其他人知道你有这个相当“有趣”的文件，不过由于你有设置适当的权限， 所以其他人自然也就无法知道该文件的内容啰！

2. 群组概念

   那么群组呢？为何要设置文件还有所属的群组？其实，群组最有用的功能之一，就是当你在团队开发资源的时候啦！ 举例来说，假设有两组专题生在我的主机里面，第一个专题组别为projecta，里面的成员有 class1, class2, class3三个；第二个专题组别为projectb，里面的成员有class4, class5, class6。 这两个专题之间是有竞争性质的，但却要缴交同一份报告。每组的组员之间必须要能够互相修改对方的数据， 但是其他组的组员则不能看到本组自己的文件内容，此时该如何是好？

   在Linux下面这样的限制是很简单啦！我可以经由简易的文件权限设置，就能限制非自己团队（亦即是群组啰） 的其他人不能够阅览内容啰！而且亦可以让自己的团队成员可以修改我所创建的文件！ 同时，如果我自己还有私人隐密的文件，仍然可以设置成让自己的团队成员也看不到我的文件数据。 很方便吧！

   另外，如果teacher这个帐号是projecta与projectb这两个专题的老师， 他想要同时观察两者的进度，因此需要能够进入这两个群组的权限时，你可以设置teacher这个帐号， “同时支持projecta与projectb这两个群组！”，也就是说：每个帐号都可以有多个群组的支持呢！

   这样说或许你还不容易理解这个使用者与群组的关系吧？没关系，我们可以使用目前“家庭”的观念来进行解说喔！ 假设有一家人，家里只有三兄弟，分别是王大毛、王二毛与王三毛三个人， 而这个家庭是登记在王大毛的名下的！所以，“王大毛家有三个人，分别是

王大毛、王二毛与王三毛"，而且这三个人都有自己的房间，并且共同拥有一个客厅喔！

- 使用者的意义：由于王家三人各自拥有自己的房间，所以，王二毛虽然可以进入王三毛的房间，但是二毛不能翻三毛的抽屉喔！那样会被三毛K的！因为抽屉里面可能有三毛自己私人的东西，例如情书啦，日记啦等等的，这是"私人的空间"，所以当然不能让二毛拿啰！

- 群组的概念：由于共同拥有客厅，所以王家三兄弟可以在客厅打开电视机啦、翻阅报纸啦、坐在沙发上面发呆啦等等的！反正，只要是在客厅的玩意儿，三兄弟都可以使用喔！因为大家都是一家人嘛！

  这样说来应该有点晓得了喔！那个"王大毛家"就是所谓的"群组"啰，至于三兄弟就是分别为三个"使用者"，而这三个使用者是在同一个群组里面的喔！而三个使用者虽然在同一群组内，但是我们可以设置"权限"，好让某些使用者个人的信息不被群组的拥有者查询，以保有个人"私人的空间"啦！而设置群组共享，则可让大家共同分享喔！

3. 其他人的概念

好了，那么今天又有个人，叫做张小猪，他是张小猪家的人，与王家没有关系啦！这个时候，除非王家认识张小猪，然后开门让张小猪进来王家，否则张小猪永远没有办法进入王家，更不要说进到王三毛的房间啦！不过，如果张小猪通过关系认识了三毛，并且跟王三毛成为好朋友，那么张小猪就可以通过三毛进入王家啦！呵呵！没错！那个张小猪就是所谓的"其他人，Others"啰！

因此，我们就可以知道啦，在Linux里面，任何一个文件都具有"User, Group及Others"三种身份的个别权限，我们可以将上面的说明以下面的图示来解释：

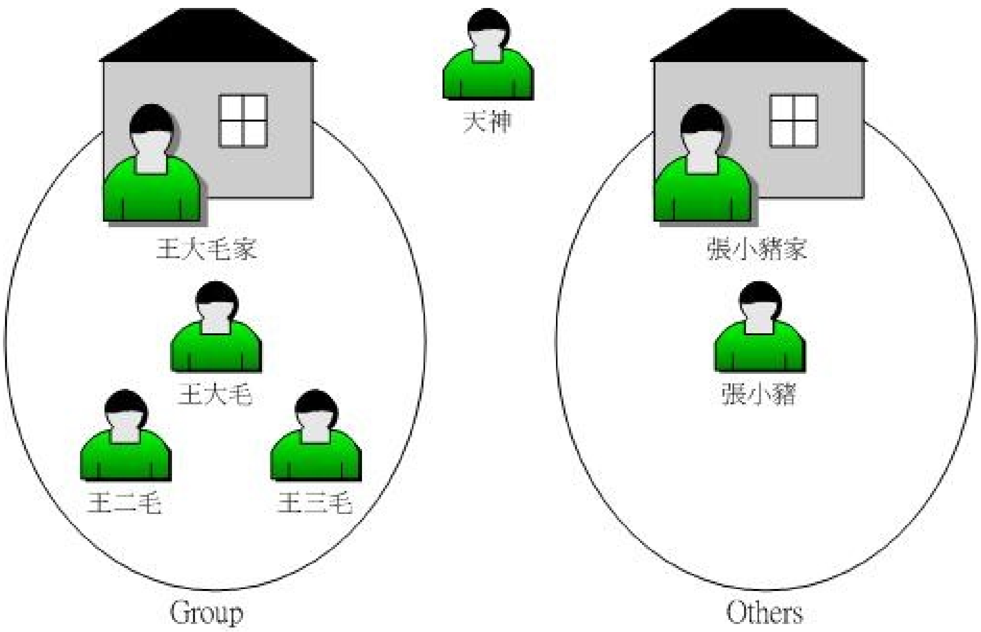


图5.1.1、每个文件的拥有者、群组与 others 的示意图我们以王三毛为例，王三毛这个"文件"的拥有者为王三毛，他属于王大毛这个群组，而张小猪相对于王三毛，则只是一个"其他人

（others）”而已。

不过，这里有个特殊的人物要来介绍的，那就是“万能的天神”！这个天神具有无限的神力， 所以他可以到达任何他想要去的地方，呵呵！那个人在Linux系统中的身份代号是“ root ”啦！所以要小心喔！那个root可是“万能的天神”喔！

无论如何，“使用者身份”，与该使用者所支持的“群组”概念，在Linux的世界里面是相当的重要的， 他可以帮助你让你的多任务Linux环境变的更容易管理！更详细的 “身份与群组” 设置，我们将在第十三章、帐号管理再进行解说。 下面我们将针对文件系统与文件权限来进行说明。



Tips 现在 （2015年） 鸟哥常以台湾地区常见的社群网站 Facebook 或者是 Google+ 作为解释。（1）你在 FB 注册一个帐号，这个帐号可以叠代对比为 Linux 的帐号， （2）你可以新增一个社团，这个社团的隐私权是可以由您自己指定的！看是要公开还是要隐藏。这就可以叠代为 Linux 的群组概念，这个群组的权限可以自己设置。 （3）那么其他在 FB 注册的人，没有加入你的社团，他就是 Linux 上所谓的“其他人”！最后，在 FB 上面的每一条留言，就可以想成 Linux 下面的“文件”啰！



Tips 那么上面内文谈到的群组有啥帮助呢？想想看，你在 FB 上面，你的 StudyArea 社团是隐藏的，你想让 dmtsai 可以进来读取每一个留言 （想成是 file） ， 最简单的作法是什么？对！让 dmstai 加入这个社团即可！没错！只要让 Linux 某个帐号加入某个群组，该帐号就可以使用该群组能够存取的资源！ 每个帐号可以加入的群组个数基本上是没有限制的！

- Linux 使用者身份与群组记录的文件

在我们Linux系统当中，默认的情况下，所有的系统上的帐号与一般身份使用者，还有那个root的相关信息， 都是记录在/etc/passwd这个文件内的。至于个人的密码则是记录在/etc/shadow这个文件下。 此外，Linux所有的群组名称都纪录在/etc/group内！这三个文件可以说是Linux系统里面帐号、密码、群组信息的集中地啰！ 不要随便删除这三个文件啊！ ^_^

至于更多的与帐号群组有关的设置，还有这三个文件的格式，不要急，我们在第十三章的帐号管理时，会再跟大家详细的介绍的！这里先有概念即可。

5.2 Linux 文件权限概念

大致了解了Linux的使用者与群组之后，接着下来，我们要来谈一谈，这个文件的权限要如何针对这些所谓的“使用者”与“群组”来设置呢？ 这个部分是相当重要的，尤其对于初学者来说，因为文件的权限与属性是学习Linux的一个相当重要的关卡， 如果没有这部份的概念，那么你将老是听不懂别人在讲什么呢！尤其是当你在你的屏幕前面出现了“Permission deny”的时候，不要担心，“肯定是权限设置错误”啦！呵呵！好了，闲话不多聊，赶快来瞧一瞧先。

5.2.1 Linux文件属性

嗯！既然要让你了解Linux的文件属性，那么有个重要的也是常用的指令就必须要先跟你说啰！那一个？就是“ ls ”这一个察看文件的指令啰！在你以dmtsai登陆系统，然后使用 su - 切换身份成为root后， 下达“ ls -al ”看看，会看到下面的几个咚咚：

```
[dmtsai@study ~]$ su -  # 先来切换一下身份看看
Password:
Last login: Tue Jun  2 19:32:31 CST 2015 on tty2
[root@study ~]# ls -al
total 48
dr-xr-x---.  5    root     root    4096  May 29 16:08 .
dr-xr-xr-x. 17    root     root    4096  May  4 17:56 ..
-rw-------.  1    root     root    1816  May  4 17:57 anaconda-ks.cfg
-rw-------.  1    root     root     927  Jun  2 11:27 .bash_history
-rw-r--r--.  1    root     root      18  Dec 29  2013 .bash_logout
-rw-r--r--.  1    root     root     176  Dec 29  2013 .bash_profile
-rw-r--r--.  1    root     root     176  Dec 29  2013 .bashrc
drwxr-xr-x.  3    root     root      17  May  6 00:14 .config             &lt;=范例说明处
drwx------.  3    root     root      24  May  4 17:59 .dbus
-rw-r--r--.  1    root     root    1864  May  4 18:01 initial-setup-ks.cfg  &lt;=范例说明处
[    1    ][  2 ][   3  ][  4 ][    5   ][     6     ][       7          ]
[  权限    ][链接][拥有者][群组][文件大小][ 修改日期 ] [       文件名        ]
```



Tips 由于本章后续的chgrp, chown等指令可能都需要使用root的身份才能够处理，所以这里建议您以root的身份来学习！要注意的是， 我们还是不建议你直接使用 root 登陆系统，建议使用 su - 这个指令来切换身份喔！离开 su - 则使用 exit 回到 dmtsai 的身份即可！

ls是“list”的意思，重点在显示文件的文件名与相关属性。而选项“-al”则表示列出所有的文件详细的权限与属性 （包含隐藏文件，就是文件名第一个字符为“ . ”的文件）。如上所示，在你第一次以root身份登陆Linux时， 如果你输入上述指令后，应该有上列的几个东西，先解释一下上面七个字段个别的意思：

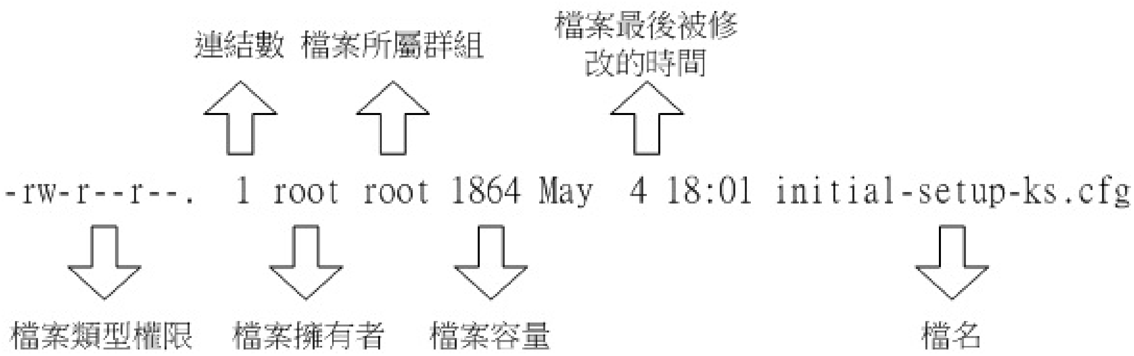


图5.2.1、文件属性的示意图

- 第一栏代表这个文件的类型与权限（permission）：

这个地方最需要注意了！仔细看的话，你应该可以发现这一栏其实共有十个字符：（图5.2.1及图5.2.2内的权限并无关系）

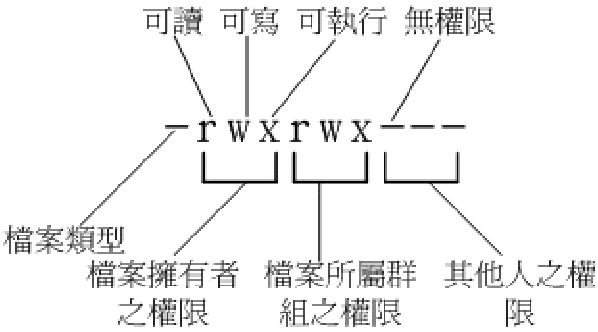


图5.2.2、文件的类型与权限之内容

- 第一个字符代表这个文件是“目录、文件或链接文件等等”：
    - 当为[d]则是目录，例如上表文件名为“.config”的那一行；
    - 当为[-]则是文件，例如上表文件名为“initial-setup-ks.cfg”那一行；
    - 若是[l]则表示为链接文件（link file）；
    - 若是[b]则表示为设备文件里面的可供储存的周边设备（可随机存取设备）；
    - 若是[c]则表示为设备文件里面的序列埠设备，例如键盘、鼠标（一次性读取设备）。
- 接下来的字符中，以三个为一组，且均为“rwx” 的三个参数的组合。其中，[r]代表可读（read）、[w]代表可写（write）、[x]代表可执行（execute）。 要注意的是，这三个权限的位置不会改变，如果没有权限，就会出现减号[-]而已。
    - 第一组为“文件拥有者可具备的权限”，以“initial-setup-ks.cfg”那个文件为例， 该文件的拥有者可以读写，但不可执行；
    - 第二组为“加入此群组之帐号的权限”；
    - 第三组为“非本人且没有加入本群组之其他帐号的权限”。



Tips 请你特别注意喔！不论是那一组权限，基本上，都是“针对某些帐号来设计的权限”喔！以群组来说，他规范的是“加入这个群组的帐号具有什么样的权限”之意， 以学校社团为例，假设学校有个童军社的社团办公室，“加入童军社的同学就可以进出社办”，主角是“学生（帐号）”而不是童军社本身喔！这样可以理解吗？

例题：若有一个文件的类型与权限数据为“-rwxr-xr--”，请说明其意义为何？答：先将整个类型与权限数据分开查阅，并将十个字符整理成为如下所示：

> [-][rwx][r-x][r--] > 1 234 567 890

1 为：代表这个文件名为目录或文件，本例中为文件（-）；234为：拥有者的权限，本例中为可读、可写、可执行（rwx）；567为：同群组使用者权限，本例中为可读可执行（rx）；890为：其他使用者权限，本例中为可读（r），就是只读之意

同时注意到，rwx所在的位置是不会改变的，有该权限就会显示字符，没有该权限就变成减号（-）就是了。

另外，目录与文件的权限意义并不相同，这是因为目录与文件所记录的数据内容不相同所致。 由于目录与文件的权限意义非常的重要，所以鸟哥将他独立到5.2.3节中的目录与文件之权限意义中再来谈。

- 第二栏表示有多少文件名链接到此节点（i-node）：

每个文件都会将他的权限与属性记录到文件系统的i-node中，不过，我们使用的目录树却是使用文件名来记录， 因此每个文件名就会链接到一个i-node啰！这个属性记录的，就是有多少不同的文件名链接到相同的一个i-node号码去就是了。 关于i-node的相关数据我们会在第七章谈到文件系统时再加强介绍的。

- 第三栏表示这个文件（或目录）的“拥有者帐号”

- 第四栏表示这个文件的所属群组

在Linux系统下，你的帐号会加入于一个或多个的群组中。举刚刚我们提到的例子，class1, class2, class3均属于projecta这个群组，假设某个文件所属的群组为projecta，且该文件的权限如图5.2.2所示（-rwxrwx---）， 则class1, class2, class3三人对于该文件都具有可读、可写、可执行的权限（看群组权限）。 但如果是不属于projecta的其他帐号，对于此文件就不具有任何权限了。

- 第五栏为这个文件的容量大小，默认单位为Bytes；

- 第六栏为这个文件的创建日期或者是最近的修改日期：

这一栏的内容分别为日期（月/日）及时间。如果这个文件被修改的时间距离现在太久了，那么时间部分会仅显示年份而已。 如下所示：

```
[root@study ~]# ll /etc/services /root/initial-setup-ks.cfg
-rw-r--r--. 1 root root 670293 Jun  7  2013 /etc/services
-rw-r--r--. 1 root root   1864 May  4 18:01 /root/initial-setup-ks.cfg
# 如上所示，/etc/services 为 2013 年所修改过的文件，离现在太远之故，所以只显示年份；
# 至于 /root/initial-setup-ks.cfg 是今年 （2015） 所创建的，所以就显示完整的时间了。
```

如果想要显示完整的时间格式，可以利用ls的选项，亦即：“ls -l --full-time”就能够显示出完整的时间格式了！包括年、月、日、时间喔。 另外，如果你当初是以繁体中文安装你的Linux系统，那么日期字段将会以中文来显示。 可惜的是，中文并没有办法在纯文本的终端机模式中正确的显示，所以此栏会变成乱码。 那你就得要使用“export LC_ALL=en_US.utf8”来修改语系喔！

如果想要让系统默认的语系变成英文的话，那么你可以修改系统配置文件“/etc/locale.conf”，利用第四章谈到的nano来修改该文件的内容，使LANG这个变量成为上述的内容即可。

- 第七栏为这个文件的文件名

这个字段就是文件名了。比较特殊的是：如果文件名之前多一个“ . ”，则代表这个文件为“隐藏文件”，例如上表中的.config那一行，该文件就是隐藏文件。 你可以使用“ls”及“ls -a”这两个指令去感受一下什么是隐藏文件啰！



Tips 对于更详细的 ls 用法，还记得怎么查询吗？对啦！使用 ls --help 或 man ls 或 info ls 去看看他的基础用法去！自我进修是很重要的， 因为“师傅带进门，修行看个人！”，自古只有天才学生，没有明星老师呦！加油吧！^_^

这七个字段的意义是很重要的！务必清楚的知道各个字段代表的意义！尤其是第一个字段的九个权限， 那是整个Linux文件权限的重点之一。下面我们来做几个简单的练习，你就会比较清楚啰！

例题：假设test1, test2, test3同属于testgroup这个群组，如果有下面的两个文件，请说明两个文件的拥有者与其相关的权限为何？

```
-rw-r--r--  1 root     root          238 Jun 18 17:22 test.txt
-rwxr-xr--  1 test1    testgroup    5238 Jun 19 10:25 ping_tsai
```

答：

- 文件test.txt的拥有者为root，所属群组为root。至于权限方面则只有root这个帐号可以存取此文件，其他人则仅能读此文件；
- 另一个文件ping_tsai的拥有者为test1，而所属群组为testgroup。其中：

  - test1 可以针对此文件具有可读可写可执行的权力；
  - 而同群组的test2, test3两个人与test1同样是testgroup的群组帐号，则仅可读可执行但不能写（亦即不能修改）；
  - 至于没有加入testgroup这一个群组的其他人则仅可以读，不能写也不能执行！

例题：承上一题如果我的目录为下面的样式，请问testgroup这个群组的成员与其他人（others）是否可以进入本目录？

```
drwxr-xr--   1 test1    testgroup    5238 Jun 19 10:25 groups/
```

答：

- 文件拥有者test1[rwx]可以在本目录中进行任何工作；
- 而testgroup这个群组[r-x]的帐号，例如test2, test3亦可以进入本目录进行工作，但是不能在本目录下进行写入的动作；
- 至于other的权限中[r--]虽然有r ，但是由于没有x的权限，因此others的使用者，并不能进入此目录！

- Linux文件权限的重要性：

与Windows系统不一样的是，在Linux系统当中，每一个文件都多加了很多的属性进来，尤其是群组的概念，这样有什么用途呢？ 其实，最大的用途是在“数据安全性”上面的。

- 系统保护的功能： 举个简单的例子，在你的系统中，关于系统服务的文件通常只有root才能读写或者是执行，例如/etc/shadow这一个帐号管理的文件，由于该文件记录了你系统中所有帐号的数据， 因此是很重要的一个配置文件，当然不能让任何人读取（否则密码会被窃取啊），只有root才能够来读取啰！所以该文件的权限就会成为[----------]啰！咦！所有人都不能使用？没关系，root基本上是不受系统的权限所限制的， 所以无论文件权限为何，默认root都可以存取喔！

- 团队开发软件或数据共享的功能： 此外，如果你有一个软件开发团队，在你的团队中，你希望每个人都可以使用某一些目录下的文件， 而非你的团队的其他人则不予以开放呢？以上面的例子来说，testgroup的团队共有三个人，分别是test1, test2, test3，那么我就可以将团队所需的文件权限订为[-rwxrws---]来提供给testgroup的工作团队使用啰！（怎么会有 s 呢？没关系，这个我们在后续章节再讲给你听！）

- 未将权限设置妥当的危害： 再举个例子来说，如果你的目录权限没有作好的话，可能造成其他人都可以在你的系统上面乱搞啰！ 例如本来只有root才能做的开关机、ADSL的拨接程序、新增或删除使用者等等的指令，若被你改成任何人都可以执行的话， 那么如果使用者不小心给你重新开机啦！重新拨接啦！等等的！那么你的系统不就会常常莫名其妙的挂掉啰！ 而且万一你的使用者的密码被其他不明人士取得的话，只要他登陆你的系统就可以轻而易举的执行一些root的工作！

可怕吧！因此，在你修改你的linux文件与目录的属性之前，一定要先搞清楚， 什么数据是可变的，什么是不可变的！千万注意啰！接下来我们来处理一下文件属性与权限的变更吧！

5.2.2 如何改变文件属性与权限

我们现在知道文件权限对于一个系统的安全重要性了，也知道文件的权限对于使用者与群组的相关性， 那么如何修改一个文件的属性与权限呢？又！有多少文件的权限我们可以修改呢？ 其实一个文件的属性与权限有很多！我们先介绍几个常用于群组、拥有者、各种身份的权限之修改的指令，如下所示：

- chgrp ：改变文件所属群组
- chown ：改变文件拥有者
- chmod ：改变文件的权限, SUID, SGID, SBIT等等的特性

- 改变所属群组, chgrp

改变一个文件的群组真是很简单的，直接以chgrp来改变即可，咦！这个指令就是change group的缩写嘛！这样就很好记了吧！ ^_^。不过，请记得，要被改变的群组名称必须要在/etc/group文件内存在才行，否则就会显示错误！

假设你已经是root的身份了，那么在你的主文件夹内有一个名为 initial-setup-ks.cfg 的文件，如何将该文件的群组改变一下呢？假设你已经知道在/etc/group里面已经存在一个名为users的群组， 但是testing这个群组名字就不存在/etc/group当中了，此时改变群组成为users与testing分别会有什么现象发生呢？

```
[root@study ~]# chgrp [-R] dirname/filename ...
选项与参数：
-R : 进行递回（recursive）的持续变更，亦即连同次目录下的所有文件、目录
     都更新成为这个群组之意。常常用在变更某一目录内所有的文件之情况。
范例：
[root@study ~]# chgrp users initial-setup-ks.cfg
[root@study ~]# ls -l
-rw-r--r--. 1 root users 1864 May  4 18:01 initial-setup-ks.cfg
[root@study ~]# chgrp testing initial-setup-ks.cfg
chgrp: invalid group:  `testing' &lt;== 发生错误讯息啰～找不到这个群组名～
```

发现了吗？文件的群组被改成users了，但是要改成testing的时候， 就会发生错误～注意喔！发生错误讯息还是要努力的查一查错误讯息的内容才好！ 将他英文翻译成为中文，就知道问题出在哪里了。

- 改变文件拥有者, chown

如何改变一个文件的拥有者呢？很简单呀！既然改变群组是change group，那么改变拥有者就是change owner啰！BINGO！那就是chown这个指令的用途，要注意的是， 使用者必须是已经存在系统中的帐号，也就是在/etc/passwd 这个文件中有纪录的使用者名称才能改变。

chown的用途还满多的，他还可以顺便直接修改群组的名称呢！此外，如果要连目录下的所有次目录或文件同时更改文件拥有者的话，直接加上 -R 的选项即可！我们来看看语法与范例：

```
[root@study ~]# chown [-R] 帐号名称 文件或目录
[root@study ~]# chown [-R] 帐号名称:群组名称 文件或目录
选项与参数：
-R : 进行递回（recursive）的持续变更，亦即连同次目录下的所有文件都变更

范例：将 initial-setup-ks.cfg 的拥有者改为bin这个帐号：
[root@study ~]# chown bin initial-setup-ks.cfg
[root@study ~]# ls -l
-rw-r--r--. 1 bin  users 1864 May  4 18:01 initial-setup-ks.cfg

范例：将 initial-setup-ks.cfg 的拥有者与群组改回为root：
[root@study ~]# chown root:root initial-setup-ks.cfg
[root@study ~]# ls -l
-rw-r--r--. 1 root root 1864 May  4 18:01 initial-setup-ks.cfg
```



Tips 事实上，chown也可以使用“**chown user.group file**”，亦即在拥有者与群组间加上小数点“ . ”也行！ 不过很多朋友设置帐号时，喜欢在帐号当中加入小数点（例如vbird.tsai这样的帐号格式），这就会造成系统的误判了！ 所以我们比较建议使用冒号“:”来隔开拥有者与群组啦！此外，chown也能单纯的修改所属群组呢！ 例如“**chown .sshd initial-setup-ks.cfg**”就是修改群组～看到了吗？就是那个小数点的用途！

知道如何改变文件的群组与拥有者了，那么什么时候要使用chown或chgrp呢？或许你会觉得奇怪吧？ 是的，确实有时候需要变更文件的拥有者的，最常见的例子就是在复制文件给你之外的其他人时， 我们使用最简单的cp指令来说明好了：

```
[root@study ~]# cp 来源文件 目的文件
```

假设你今天要将.bashrc这个文件拷贝成为.bashrc_test文件名，且是要给bin这个人，你可以这样做：

```
[root@study ~]# cp .bashrc .bashrc_test
[root@study ~]# ls -al .bashrc*
-rw-r--r--. 1 root root 176 Dec 29  2013 .bashrc
-rw-r--r--. 1 root root 176 Jun  3 00:04 .bashrc_test   &lt;==新文件的属性没变
```

由于复制行为（cp）会复制执行者的属性与权限，所以！怎么办？.bashrc_test还是属于root所拥有， 如此一来，即使你将文件拿给bin这个使用者了，那他仍然无法修改的（看属性/权限就知道了吧）， 所以你就必须要将这个文件的拥有者与群组修改一下啰！知道如何修改了吧？

- 改变权限, chmod

文件权限的改变使用的是chmod这个指令，但是，权限的设置方法有两种， 分别可以使用数字或者是符号来进行权限的变更。我们就来谈一谈：

- 数字类型改变文件权限

Linux文件的基本权限就有九个，分别是owner/group/others三种身份各有自己的read/write/execute权限， 先复习一下刚刚上面提到的数据：文件的权限字符为：“-rwxrwxrwx”， 这九个权限是三个三个一组的！其中，我们可以使用数字来代表各个权限，各权限的分数对照表如下：

> r:4 > w:2 > x:1

每种身份（owner/group/others）各自的三个权限（r/w/x）分数是需要累加的，例如当权限为： [-rwxrwx---] 分数则是：

> owner = rwx = 4+2+1 = 7 > group = rwx = 4+2+1 = 7 > others= --- = 0+0+0 = 0

所以等一下我们设置权限的变更时，该文件的权限数字就是770啦！变更权限的指令chmod的语法是这样的：

```
[root@study ~]# chmod [-R] xyz 文件或目录
选项与参数：
xyz : 就是刚刚提到的数字类型的权限属性，为 rwx 属性数值的相加。
-R : 进行递回（recursive）的持续变更，亦即连同次目录下的所有文件都会变更
```

举例来说，如果要将.bashrc这个文件所有的权限都设置启用，那么就下达：

```
[root@study ~]# ls -al .bashrc
-rw-r--r--. 1 root root 176 Dec 29  2013 .bashrc
[root@study ~]# chmod 777 .bashrc
[root@study ~]# ls -al .bashrc
-rwxrwxrwx. 1 root root 176 Dec 29  2013 .bashrc
```

那如果要将权限变成“ -rwxr-xr-- ”呢？那么权限的分数就成为 [4+2+1][4+0+1][4+0+0]=754 啰！所以你需要下达“ chmod 754 filename”。 另外，在实际的系统运行中最常发生的一个问题就是，常常我们以vim编辑一个shell的文字批处理文件后，他的权限通常是 -rw-rw-r-- 也就是664， 如果要将该文件变成可执行文件，并且不要让其他人修改此一文件的话， 那么就需要-rwxr-xr-x这样的权限，此时就得要下达：“ chmod 755 test.sh ”的指令啰！

另外，如果有些文件你不希望被其他人看到，那么应该将文件的权限设置为例如：“-rwxr-----”，那就下达“ chmod 740 filename ”吧！

例题：将刚刚你的.bashrc这个文件的权限修改回-rw-r--r--的情况吧！答：-rw-r--r--的分数是644，所以指令为： `chmod 644 .bashrc`

- 符号类型改变文件权限

还有一个改变权限的方法呦！从之前的介绍中我们可以发现，基本上就九个权限分别是（1）user （2）group （3）others三种身份啦！那么我们就可以借由u, g, o来代表三种身份的权限！此外， a 则代表 all 亦即全部的身份！那么读写的权限就可以写成r, w, x

啰！也就是可以使用下面的方式来看：

```
| chmod | u g o a | +（加入） -（除去） =（设置） | r w x | 文件或目录 |
```

来实作一下吧！假如我们要“设置”一个文件的权限成为“-rwxr-xr-x”时，基本上就是：

- user （u）：具有可读、可写、可执行的权限；
- group 与 others （g/o）：具有可读与执行的权限。 所以就是：

```
[root@study ~]# chmod  u=rwx,go=rx  .bashrc
# 注意喔！那个 u=rwx,go=rx 是连在一起的，中间并没有任何空白字符！
[root@study ~]# ls -al .bashrc
-rwxr-xr-x. 1 root root 176 Dec 29  2013 .bashrc
```

那么假如是“ -rwxr-xr-- ”这样的权限呢？可以使用“ chmod u=rwx,g=rx,o=r filename ”来设置。此外，如果我不知道原先的文件属性，而我只想要增加.bashrc这个文件的每个人均可写入的权限， 那么我就可以使用：

```
[root@study ~]# ls -al .bashrc
-rwxr-xr-x. 1 root root 176 Dec 29  2013 .bashrc
[root@study ~]# chmod  a+w  .bashrc
[root@study ~]# ls -al .bashrc
-rwxrwxrwx. 1 root root 176 Dec 29  2013 .bashrc
```

而如果是要将权限去掉而不更动其他已存在的权限呢？例如要拿掉全部人的可执行权限，则：s

```
[root@study ~]# chmod  a-x  .bashrc
[root@study ~]# ls -al .bashrc
-rw-rw-rw-. 1 root root 176 Dec 29  2013 .bashrc
[root@study ~]# chmod 644 .bashrc  # 测试完毕得要改回来喔！
```

知道 +, -, = 的不同点了吗？对啦！ + 与 – 的状态下，只要是没有指定到的项目，则该权限“不会被变动”， 例如上面的例子中，由于仅以 – 拿掉 x 则其他两个保持当时的值不变！多多实作一下，你就会知道如何改变权限啰！ 这在某些情况下面很好用的～举例来说，你想要教一个朋友如何让一个程序可以拥有执行的权限， 但你又不知道该文件原本的权限为何，此时，利用“chmod a+x filename” ，就可以让该程序拥有执行的权限了。是否很方便？

5.2.3 目录与文件之权限意义：

现在我们知道了Linux系统内文件的三种身份（拥有者、群组与其他人），知道每种身份都有三种权限（rwx）， 已知道能够使用chown, chgrp, chmod去修改这些权限与属性，当然，利用ls -l去观察文件也没问题。 前两小节也谈到了这些文件权限对于数据安全的重要性。那么，这些文件权限对于一般文件与目录文件有何不同呢？ 有大大的不同啊！下面就让鸟哥来说清楚，讲明白！

- 权限对文件的重要性

文件是实际含有数据的地方，包括一般文本文件、数据库内容档、二进制可可执行文件（binary program）等等。 因此，权限对于文件来说，他的意义是这样的：

- r （read）：可读取此一文件的实际内容，如读取文本文件的文字内容等；
- w （write）：可以编辑、新增或者是修改该文件的内容（但不含删除该文件）；
- x （eXecute）：该文件具有可以被系统执行的权限。

那个可读（r）代表读取文件内容是还好了解，那么可执行（x）呢？这里你就必须要小心啦！因为在Windows下面一个文件是否具有执行的能力是借由“ 扩展名 ”来判断的， 例如：.exe, .bat, .com 等等，但是在Linux下面，我们的文件是否能被执行，则是借由是否具有“x”这个权限来决定的！跟文件名是没有绝对的关系的！

至于最后一个w这个权限呢？当你对一个文件具有w权限时，你可以具有写入/编辑/新增/修改文件的内容的权限， 但并不具备有删除该文件本身的权限！对于文件的rwx来说， 主要都是针对“文件的内容”而言，与文件文件名的存在与否没有关系喔！因为文件记录的是实际的数据嘛！

- 权限对目录的重要性

文件是存放实际数据的所在，那么目录主要是储存啥玩意啊？目录主要的内容在记录文件名清单，文件名与目录有强烈的关连啦！ 所以如果是针对目录时，那个 r, w, x 对目录是什么意义呢？

- r （read contents in directory）：

  表示具有读取目录结构清单的权限，所以当你具有读取（r）一个目录的权限时，表示你可以查询该目录下的文件名数据。 所以你就可以利用 ls 这个指令将该目录的内容列表显示出来！

- w （modify contents of directory）：

  这个可写入的权限对目录来说，是很了不起的！ 因为他表示你具有异动该目录结构清单的权限，也就是下面这些权限：

  - 创建新的文件与目录；
  - 删除已经存在的文件与目录（不论该文件的权限为何！）
  - 将已存在的文件或目录进行更名；
  - 搬移该目录内的文件、目录位置。 总之，目录的w权限就与该目录下面的文件名异动有关就对了啦！

- x （access directory）：

  咦！目录的执行权限有啥用途啊？目录只是记录文件名而已，总不能拿来执行吧？没错！目录不可以被执行，目录的x代表的是使用者能否进入该目录成为工作目录的用途！所谓的工作目录（work directory）就是你目前所在的目录啦！举例来说，当你登陆Linux

时， 你所在的主文件夹就是你当下的工作目录。而变换目录的指令是“cd”（change directory）啰！

上面的东西这么说，也太条列式～太教条了～有没有清晰一点的说明啊？好～让我们来思考一下人类社会使用的东西好了！ 现在假设“文件是一堆文件数据夹”，所以你可能可以在上面写/改一些数据。而“目录是一堆抽屉”，因此你可以将数据夹分类放置到不同的抽屉去。 因此抽屉最大的目的是拿出/放入数据夹喔！现在让我们汇整一下数据：

元件	内容	叠代物件	r	w	x
文件	详细数据data	文件数据夹	读到文件内容	修改文件内容	执行文件内容
目录	文件名	可分类抽屉	读到文件名	修改文件名	进入该目录的权限（key）

根据上述的分析，你可以看到，对一般文件来说，rwx 主要是针对“文件的内容”来设计权限，对目录来说，rwx则是针对“目录内的文件名列表”来设计权限。 其中最有趣的大概就属目录的 x 权限了！“文件名怎么执行”？没道理嘛！其实，这个 x 权限设计，就相当于“该目录，也就是该抽屉的 "钥匙" ”啦！ 没有钥匙你怎么能够打开抽屉呢？对吧！

大致的目录权限概念是这样，下面我们来看几个范例，让你了解一下啥是目录的权限啰！

例题：有个目录的权限如下所示：

```
drwxr--r--  3  root  root  4096   Jun 25 08:35   .ssh
```

系统有个帐号名称为vbird，这个帐号并没有支持root群组，请问vbird对这个目录有何权限？是否可切换到此目录中？答：vbird对此目录仅具有r的权限，因此vbird可以查询此目录下的文件名列表。因为vbird不具有x的权限，亦即 vbird 没有这个抽屉的钥匙啦！ 因此vbird并不能切换到此目录内！（相当重要的概念！） |

上面这个例题中因为vbird具有r的权限，因为是r乍看之下好像就具有可以进入此目录的权限，其实那是错的。 能不能进入某一个目录，只与该目录的x权限有关啦！此外， 工作目录对于指令的执行是非常重要的，如果你在某目录下不具有x的权限， 那么你就无法切换到该目录下，也就无法执行该目录下的任何指令，即使你具有该目录的r或w的权限。

很多朋友在架设网站的时候都会卡在一些权限的设置上，他们开放目录数据给网际网络的任何人来浏览， 却只开放r的权限，如上面的范例所示那样，那样的结果就是导致网站服务器软件无法到该目录下读取文件（最多只能看到文件名）， 最终用户总是无法正确的查阅到文件的内容（显示权限不足啊！）。要注意：要开放目录给任何人浏览时，应该至少也要给予r及x的权限，但w权限不可随便给！ 为什么w不能随便给，我们来看下一个例子：

例题：假设有个帐号名称为dmtsai，他的主文件夹在/home/dmtsai/，dmtsai对此目录具有[rwx]的权限。 若在此目录下有个名为the_root.data的文件，该文件的权限如下：

```
-rwx------ 1 root  root  4365 Sep 19 23:20  the_root.data
```

请问dmtsai对此文件的权限为何？可否删除此文件？答：如上所示，由于dmtsai对此文件来说是“others”的身份，因此这个文件他无法读、无法编辑也无法执行， 也就是说，他无法变动这个文件的内容就是了。

但是由于这个文件在他的主文件夹下， 他在此目录下具有rwx的完整权限，因此对于the_root.data这个“文件名”来说，他是能够“删除”的！ 结论就是，dmtsai这个用户能够删除the_root.data这个文件！ |



Tips 上述的例子解释是这样的，假设有个莫名其妙的人，拿着一个完全密封的数据夹放到你的办公室抽屉中，因为完全密封你也打不开、看不到这个数据夹的内部数据（对文件来说，你没有权限）。 但是因为这个数据夹是放在你的抽屉中，你当然可以拿出/放入任何数据在这个抽屉中（对目录来说，你具有所有权限）。 所以，情况就是：你打开抽屉、拿出这个没办法看到的数据夹、将他丢到走廊上的垃圾桶！搞定了 （顺利删除！）！

还是看不太懂？有听没有懂喔！没关系～我们下面就来设计一个练习， 让你实际玩玩看，应该就能够比较近入状况啦！不过，由于很多指令我们还没有教， 所以下面的指令有的先了解即可，详细的指令用法我们会在后面继续介绍的。

- 先用root的身份创建所需要的文件与目录环境

我们用root的身份在所有人都可以工作的/tmp目录中创建一个名为testing的目录， 该目录的权限为744且目录拥有者为root。另外，在testing目录下在创建一个空的文件， 文件名亦为testing。创建目录可用mkdir（make directory），创建空文件可用touch（下一章会说明）来处理。 所以过程如下所示：

```
[root@study ~]# cd /tmp &lt;==切换工作目录到/tmp
[root@study tmp]# mkdir testing &lt;==创建新目录
[root@study tmp]# chmod 744 testing &lt;==变更权限
[root@study tmp]# touch testing/testing &lt;==创建空的文件
[root@study tmp]# chmod 600 testing/testing &lt;==变更权限
[root@study tmp]# ls -ald testing testing/testing
drwxr--r--. 2 root root 20 Jun  3 01:00 testing
-rw-------. 1 root root  0 Jun  3 01:00 testing/testing
# 仔细看一下，目录的权限是 744 ，且所属群组与使用者均是 root 喔！
# 那么在这样的情况下面，一般身份使用者对这个目录/文件的权限为何？
```

- 一般用户的读写权限为何？观察中

在上面的例子中，虽然目录是744的权限设置，一般用户应该能有 r 的权限， 但这样的权限使用者能做啥事呢？由于鸟哥的系统中含有一个帐号名为 dmtsai 的，请再开另外一个终端机，使用 dmtsai 登陆来操作下面的任务！

```
[dmtsai@study ~]$ cd /tmp
[dmtsai@study tmp]$ ls -l testing/
ls: cannot access testing/testing: Permission denied
total 0
?????????? ? ? ? ?            ? testing
# 虽然有告知权限不足，但因为具有 r 的权限可以查询文件名。由于权限不足（没有x），所以会有一堆问号。
[dmtsai@study tmp]$ cd testing/
-bash: cd: testing/: Permission denied
# 因为不具有 x ，所以当然没有进入的权限啦！有没有呼应前面的权限说明啊！
```

- 如果该目录属于用户本身，会有什么状况？

上面的练习我们知道了只有r确实可以让使用者读取目录的文件名列表，不过详细的信息却还是读不到的， 同时也不能将该目录变成工作目录（用 cd 进入该目录之意）。那如果我们让该目录变成使用者的， 那么使用者在这个目录下面是否能够删除文件呢？下面的练习做看看：

```
# 1\. 先用 root 的身份来搞定 /tmp/testing 的属性、权限设置：
[root@study tmp]# chown dmtsai /tmp/testing
[root@study tmp]# ls -ld /tmp/testing
drwxr--r--. 2 dmtsai root 20  6月  3 01:00 /tmp/testing  # dmtsai 是具有全部权限的！

# 2\. 再用 dmtsai 的帐号来处理一下 /tmp/testing/testing 这个文件看看：
[dmtsai@study tmp]$ cd /tmp/testing
[dmtsai@study testing]$ ls -l  &lt;==确实是可以进入目录
-rw-------. 1 root root 0 Jun  3 01:00 testing  &lt;==文件不是vbird的！
[dmtsai@study testing]$ rm testing     &lt;==尝试杀掉这个文件看看！
rm: remove write-protected regular empty file `testing'? y
# 竟然可以删除！这样理解了吗？！
```

通过上面这个简单的步骤，你就可以清楚的知道， x 在目录当中是与“能否进入该目录”有关， 至于那个 w 则具有相当重要的权限，因为他可以让使用者删除、更新、新建文件或目录， 是个很重要的参数啊！这样可以理解了吗？！ ^_^！

- 使用者操作功能与权限

刚刚讲这样如果你还是搞不懂～没关系，我们来处理个特殊的案例！假设两个文件名，分别是下面这样：

- /dir1/file1
- /dir2

假设你现在在系统使用 dmtsai 这个帐号，那么这个帐号针对 /dir1, /dir1/file1, /dir2 这三个文件名来说，分别需要“哪些最小的权限”才能达成各项任务？ 鸟哥汇整如下，如果你看得懂，恭喜你，如果你看不懂～没关系～未来再来继续学！

操作动作	/dir1	/dir1/file1	/dir2	重点
读取 file1 内容	x	r	-	要能够进入 /dir1 才能读到里面的文件数据！
修改 file1 内容	x	rw	-	能够进入 /dir1 且修改 file1 才行！
执行 file1 内容	x	rx	-	能够进入 /dir1 且 file1 能运行才行！
删除 file1 文件	wx	-	-	能够进入 /dir1 具有目录修改的权限即可！
将 file1 复制到 /dir2	x	r	wx	要能够读 file1 且能够修改 /dir2 内的数据

你可能会问，上面的表格当中，很多时候 /dir1 都不必有 r 耶！为啥？我们知道 /dir1 是个目录，也是个抽屉！那个抽屉的 r 代表"这个抽屉里面有灯光"， 所以你能看到的抽屉内的所有数据夹名称 （非内容）。但你已经知道里面的数据夹放在哪个地方，那，有没有灯光有差嘛？你还是可以摸黑拿到该数据夹的！对吧！ 因此，上面很多动作中，你只要具有 x 即可！r 是非必备的！只是，没有 r 的话，使用 [tab] 时，他就无法自动帮你补齐文件名了！这样理解乎？



Tips 看了上面这个表格，你应该会觉得很可怕喔！因为，要读一个文件时，你得要具有"这个文件所在目录的 x 权限"才行！所以，通常要开放的目录， 至少会具备 rx 这两个权限！现在你知道为啥了吧？

5.2.4 Linux文件种类与扩展名

我们在基础篇一直强调一个概念，那就是：任何设备在Linux下面都是文件， 不仅如此，连数据沟通的接口也有专属的文件在负责～所以，你会了解到，Linux的文件种类真的很多～ 除了前面提到的一般文件（-）与目录文件（d）之外，还有哪些种类的文件呢？

- 文件种类：

我们在刚刚提到使用"ls -l"观察到第一栏那十个字符中，第一个字符为文件的类型。 除了常见的一般文件（-）与目录文件（d）之外，还有哪些种类的文件类型呢？

- 正规文件（regular file ）： 就是一般我们在进行存取的类型的文件，在由 ls -al 所显示出来的属性方面，第一个字符为 [-]，例如 [-rwxrwxrwx]。另外，依照文件的内容，又大略可以分为：

  - 纯文本文件（ASCII）：这是Linux系统中最多的一种文件类型啰， 称为纯文本文件是因为内容为我们人类可以直接读到的数据，例如数字、字母等等。 几乎只要我们可以用来做为设置的文件都属于这一种文件类型。 举例来说，你可以下达“ cat ~/.bashrc ”就可以看到该文件的内容。 （cat 是将一个文件内容读出来的指令）

  - 二进制档（binary）：还记得我们在“ 第零章、计算机概论 ”里面的软件程序的运行中提过， 我们的系统其实仅认识且可以执行二进制文件（binary file）吧？没错～ 你的Linux当中的可可执行文件（scripts, 文字体批处理文件不算）就是这种格式的啦～ 举例来说，刚刚下达的指令cat就是一个binary file。

  - 数据格式文件（data）： 有些程序在运行的过程当中会读取某些特定格式的文件，那些特定格式的文件可以被称为数据文件 （data file）。举例来说，我们的Linux在使用者登陆时，都会将登录的数据记录在 /var/log/wtmp那个文件内，该文件是一个data file，他能够通过last这个指令读出来！ 但是使用cat时，会读出乱码～因为他是属于一种特殊格式的文件。瞭乎？

- 目录（directory）： 就是目录啰～第一个属性为 [d]，例如 [drwxrwxrwx]。

- 链接文件（link）： 就是类似Windows系统下面的捷径啦！ 第一个属性为 [l]（英文L的小写），例如 [lrwxrwxrwx] ；

- 设备与设备文件（device）： 与系统周边及储存等相关的一些文件， 通常都集中在/dev这个目录之下！通常又分为两种：

  - 区块（block）设备文件 ：就是一些储存数据， 以提供系统随机存取的周边设备，举例来说，硬盘与软盘等就是啦！ 你可以随机的在硬盘的不同区块读写，这种设备就是区块设备啰！你可以自行查一下/dev/sda看看， 会发现第一个属性为[b]喔！

  - 字符（character）设备文件：亦即是一些序列埠的周边设备， 例如键盘、鼠标等等！这些设备的特色就是“一次性读取”的，不能够截断输出。 举例来说，你不可能让鼠标“跳到”另一个画面，而是“连续性滑动”到另一个地方啊！第一个属性为 [c]。

- 数据接口文件（sockets）： 既然被称为数据接口文件， 想当然尔，这种类型的文件通常被用在网络上的数据承接了。我们可以启动一个程序来监听用户端的要求， 而用户端就可以通过这个socket来进行数据的沟通了。第一个属性为 [s]， 最常在/run或/tmp这些个目录中看到这种文件类型了。

- 数据输送档（FIFO, pipe）： FIFO也是一种特殊的文件类型，他主要的目的在解决多个程序同时存取一个文件所造成的错误问题。 FIFO是first-in-first-out的缩写。第一个属性为[p] 。

除了设备文件是我们系统中很重要的文件，最好不要随意修改之外（通常他也不会让你修改的啦！）， 另一个比较有趣的文件就是链接文件。如果你常常将应用程序捉到桌面来的话，你就应该知道在 Windows下面有所谓的“捷径”。同样的，你可以将 linux下的链接文件简单的

视为一个文件或目录的捷径。 至于socket与FIFO文件比较难理解，因为这两个咚咚与程序（process）比较有关系， 这个等到未来你了解process之后，再回来查阅吧！此外， 你也可以通过man fifo及man socket来查阅系统上的说明！

- Linux文件扩展名：

基本上，Linux的文件是没有所谓的“扩展名”的，我们刚刚就谈过，一个Linux文件能不能被执行，与他的第一栏的十个属性有关， 与文件名根本一点关系也没有。这个观念跟Windows的情况不相同喔！在Windows下面， 能被执行的文件扩展名通常是 .com .exe .bat等等，而在Linux下面，只要你的权限当中具有x的话，例如[-rwxr-xr-x] 即代表这个文件具有可以被执行的能力喔！



Tips 具有“可执行的权限”以及“具有可执行的程序码”是两回事！在 Linux 下面，你可以让一个文本文件，例如我们之前写的 text.txt 具有“可执行的权限” （加入 x 权限即可）， 但是这个文件明显的无法执行，因为他不具备可执行的程序码！而如果你将上面提到的 cat 这个可以执行的指令，将他的 x 拿掉，那么 cat 将无法被你执行！

不过，可以被执行跟可以执行成功是不一样的～举例来说，在root主文件夹下的 initial-setup-ks.cfg 是一个纯文本文件，如果经由修改权限成为 -rwxrwxrwx 后，这个文件能够真的执行成功吗？ 当然不行～因为他的内容根本就没有可以执行的数据。所以说，这个x代表这个文件具有可执行的能力， 但是能不能执行成功，当然就得要看该文件的内容啰～

虽然如此，不过我们仍然希望可以借由扩展名来了解该文件是什么东西，所以， 通常我们还是会以适当的扩展名来表示该文件是什么种类的。下面有数种常用的扩展名：

- *.sh ： 脚本或批处理文件 （scripts），因为批处理文件为使用shell写成的，所以扩展名就编成 .sh 啰；
- Z, .tar, .tar.gz, .zip, *.tgz： 经过打包的压缩文件。这是因为压缩软件分别为 gunzip, tar 等等的，由于不同的压缩软件，而取其相关的扩展名啰！
- .html, .php：网页相关文件，分别代表 HTML 语法与 PHP 语法的网页文件啰！ .html 的文件可使用网页浏览器来直接打开，至于 .php 的文件， 则可以通过 client 端的浏览器来 server 端浏览，以得到运算后的网页结果呢！

基本上，Linux系统上的文件名真的只是让你了解该文件可能的用途而已， 真正的执行与否仍然需要权限的规范才行！例如虽然有一个文件为可执行文件， 如常见的/bin/ls这个显示文件属性的指令，不过，如果这个文件的权限被修改成无法执行时， 那么ls就变成不能执行啰！

上述的这种问题最常发生在文件传送的过程中。例如你在网络上下载一个可可执行文件，但是偏偏在你的 Linux系统中就是无法执行！呵呵！那么就是可能文件的属性被改变了！不要怀疑，从网络上传送到你的 Linux系统中，文件的属性与权限确实是会被改变的喔！

- Linux文件长度限制[1]：

在Linux下面，使用传统的Ext2/Ext3/Ext4文件系统以及近来被 CentOS 7 当作默认文件系统的 xfs 而言，针对文件的文件名长度限制为：

- 单一文件或目录的最大容许文件名为 255Bytes，以一个 ASCII 英文占用一个 Bytes 来说，则大约可达 255 个字符长度。若是以每个中文字占用 2Bytes 来说， 最大文件名就是大约在 128 个中文字之谱！

是相当长的文件名喔！我们希望Linux的文件名称可以一看就知道该文件在干嘛的， 所以文件名通常是很长很长！而用惯了Windows的人可能会受不了，因为文件名称通常真的都很长，对于用惯Windows而导致打字速度不快的朋友来说，嗯！真的是很困扰.....不过，只得劝你好好的加强打字的训练啰！

- Linux文件名称的限制：

由于Linux在命令行下的一些指令操作关系，一般来说，你在设置Linux下面的文件名称时，最好可以避免一些特殊字符比较好！例如下面这些：

> - ? > < ; & ! [ ] | \ ' " ` （ ） { }

因为这些符号在命令行下，是有特殊意义的！另外，文件名称的开头为小数点“.”时， 代表这个文件为“隐藏文件”喔！同时，由于指令下达当中，常常会使用到 -option 之类的选项， 所以你最好也避免将文件文件名的开头以 - 或 + 来命名啊！

5.3 Linux目录配置

在了解了每个文件的相关种类与属性，以及了解了如何更改文件属性/权限的相关信息后，再来要了解的就是， 为什么每套Linux distributions他们的配置文件啊、可执行文件啊、每个目录内放置的咚咚啊，其实都差不多？ 原来是有一套标准依据的哩！我们下面就来瞧一瞧。

5.3.1 Linux目录配置的依据--FHS

因为利用Linux来开发产品或distributions的社群/公司与个人实在太多了，如果每个人都用自己的想法来配置文件放置的目录，那么将可能造成很多管理上的困扰。 你能想像，你进入一个企业之后，所接触到的Linux目录配置方法竟然跟你以前学的完全不同吗？很难想像吧～所以，后来就有所谓的Filesystem Hierarchy Standard （FHS）标准的出炉了！

根据FHS[2]的标准文件指出，他们的主要目的是希望让使用者可以了解到已安装软件通常放置于那个目录下， 所以他们希望独立的软件开发商、操作系统制作者、以及想要维护系统的使用者，都能够遵循FHS的标准。 也就是说，FHS的重点在于规范每个特定的目录下应该要放置什么样子的数据而已。 这样做好处非常多，因为Linux操作系统就能够在既有的面貌下（目录架构不变）发展出开发者想要的独特风格。

事实上，FHS是根据过去的经验一直再持续的改版的，FHS依据文件系统使用的频繁与否与是否允许使用者随意更动， 而将目录定义成为四种交互作用的形态，用表格来说有点像下面这样：

	可分享的（**shareable**）	不可分享的（**unshareable**）
不变的（static）	/usr （软件放置处）	/etc （配置文件）
/opt （第三方协力软件）	/boot （开机与核心档）	
可变动的（variable）	/var/mail （使用者邮件信箱）	/var/run （程序相关）
/var/spool/news （新闻群组）	/var/lock （程序相关）	

上表中的目录就是一些代表性的目录，该目录下面所放置的数据在下面会谈到，这里先略过不谈。 我们要了解的是，什么是那四个类型？

- 可分享的：可以分享给其他系统挂载使用的目录，所以包括可执行文件与使用者的邮件等数据， 是能够分享给网络上其他主机挂载用的目录；
- 不可分享的：自己机器上面运行的设备文件或者是与程序有关的socket文件等， 由于仅与自身机器有关，所以当然就不适合分享给其他主机了。

- 不变的：有些数据是不会经常变动的，跟随着distribution而不变动。 例如函数库、文件说明文档、系统管理员所管理的主机服务配置文件等等；
- 可变动的：经常改变的数据，例如登录文件、一般用户可自行收受的新闻群组等。

事实上，FHS针对目录树架构仅定义出三层目录下面应该放置什么数据而已，分别是下面这三个目录的定义：

- / （root, 根目录）：与开机系统有关；
- /usr （unix software resource）：与软件安装/执行有关；
- /var （variable）：与系统运行过程有关。

为什么要定义出这三层目录呢？其实是有意义的喔！每层目录下面所应该要放置的目录也都又特定的规定喔！ 由于我们尚未介绍完整的Linux系统，所以下面的介绍你可能会看不懂！没关系，先有个概念即可， 等到你将基础篇全部看完后，就重头将基础篇再看一遍！到时候你就会豁然开朗啦！^_^



Tips 这个 root 在 Linux 里面的意义真的很多很多～多到让人搞不懂那是啥玩意儿。 如果以“帐号”的角度来看，所谓的 root 指的是“系统管理员！”的身份， 如果以“目录”的角度来看，所谓的 root 意即指的是根目录，就是 / 啦～ 要特别留意喔！

- 根目录 （/） 的意义与内容：

根目录是整个系统最重要的一个目录，因为不但所有的目录都是由根目录衍生出来的，同时根目录也与开机/还原/系统修复等动作有关。 由于系统开机时需要特定的开机软件、核心文件、开机所需程序、函数库等等文件数据，若系统出现错误时，根目录也必须要包含有能够修复文件系统的程序才行。 因为根目录是这么的重要，所以在FHS的要求方面，他希望根目录不要放在非常大的分区内， 因为越大的分区你会放入越多的数据，如此一来根目录所在分区就可能会有较多发生错误的机会。

因此FHS标准建议：根目录（/）所在分区应该越小越好， 且应用程序所安装的软件最好不要与根目录放在同一个分区内，保持根目录越小越好。 如此不但性能较佳，根目录所在的文件系统也较不容易发生问题。

有鉴于上述的说明，因此FHS定义出根目录（/）下面应该要有下面这些次目录的存在才好，即使没有实体目录，FHS也希望至少有链接文件存在才好：

目录	应放置文件内容
第一部份：FHS	

要求必须要存在的目录	
/bin	系统有很多放置可执行文件的目录，但/bin比较特殊。因为/bin放置的是在单人维护模式下还能够被操作的指令。 在/bin下面的指令可以被root与一般帐号所使用，主要有：cat, chmod, chown, date, mv, mkdir, cp, bash等等常用的指令。
/boot	这个目录主要在放置开机会使用到的文件，包括Linux核心文件以及开机菜单与开机所需配置文件等等。 Linux kernel常用的文件名为：vmlinuz，如果使用的是grub2这个开机管理程序， 则还会存在/boot/grub2/这个目录喔！
/dev	在Linux系统上，任何设备与周边设备都是以文件的型态存在于这个目录当中的。 你只要通过存取这个目录下面的某个文件，就等于存取某个设备啰～ 比要重要的文件有/dev/null, /dev/zero, /dev/tty, /dev/loop*, /dev/sd*等等
/etc	系统主要的配置文件几乎都放置在这个目录内，例如人员的帐号密码档、各种服务的启始档等等。一般来说，这个目录下的各文件属性是可以让一般使用者查阅的， 但是只有root有权力修改。FHS建议不要放置可可执行文件（binary）在这个目录中喔。比较重要的文件有： /etc/modprobe.d/, /etc/passwd, /etc/fstab, /etc/issue 等等。另外 FHS 还规范几个重要的目录最好要存在 /etc/ 目录下喔：/etc/opt（必要）：这个目录在放置第三方协力软件 /opt 的相关配置文件 /etc/X11/（建议）：与 X Window 有关的各种配置文件都在这里，尤其是 xorg.conf 这个 X Server 的配置文件。 /etc/sgml/（建议）：与 SGML 格式有关的各项配置文件 /etc/xml/（建议）：与 XML 格式有关的各项配置文件
/lib	系统的函数库非常的多，而/lib放置的则是在开机时会用到的函数库， 以及在/bin或/sbin下面的指令会调用的函数库而已。 什么是函数库呢？你可以将他想成是“外挂”，某些指令必须要有这些“外挂”才能够顺利完成程序的执行之意。 另外 FSH 还要求下面的目录必须要存在：/lib/modules/：这个目录主要放置可抽换式的核心相关模块（驱动程序）喔！
/media	media是“媒体”的英文，顾名思义，这个/media下面放置的就是可移除的设备啦！ 包括软盘、光盘、DVD等等设备都暂时挂载于此。常见的文件名有：/media/floppy, /media/cdrom等等。
/mnt	如果你想要暂时挂载某些额外的设备，一般建议你可以放置到这个目录中。在古早时候，这个目录的用途与/media相同啦！只是有了/media之后，这个目录就用来暂时挂载用了。
/opt	这个是给第三方协力软件放置的目录。什么是第三方协力软件啊？ 举例来说，KDE这个桌面管理系统是一个独立的计划，不过他可以安装到Linux系统中，因此KDE的软件就建议放置到此目录下了。 另外，如果你想要自行安装额外的软件（非原本的distribution提供的），那么也能够将你的软件安装到这里来。 不过，以前的Linux系统中，我们还是习惯放置在/usr/local目录下呢！
/run	早期的 FHS 规定系统开机后所产生的各项信息应该要放置到 /var/run 目录下，新版的 FHS 则规范到 /run 下面。 由于 /run 可以使用内存来仿真，因此性能上会好很多！
	Linux有非常多指令是用来设置系统环境的，这些指令只有root才能够利用来“设置”系统，其他使用者最多只能用来“查询”而已。 放在/sbin下面的为开机过程中所需要的，里面包括了开机、修复、还原系统所需要的指令。 至

	于某些服务器软件程序，一般则放置到/usr/sbin/当中。至于本机自行安装的软件所产生的系统可执行文件（system binary），则放置到/usr/local/sbin/当中了。常见的指令包括：fdisk, fsck, ifconfig, mkfs等等。
/srv	srv可以视为“service”的缩写，是一些网络服务启动之后，这些服务所需要取用的数据目录。 常见的服务例如WWW, FTP等等。举例来说，WWW服务器需要的网页数据就可以放置在/srv/www/里面。 不过，系统的服务数据如果尚未要提供给网际网络任何人浏览的话，默认还是建议放置到 /var/lib 下面即可。
/tmp	这是让一般使用者或者是正在执行的程序暂时放置文件的地方。 这个目录是任何人都能够存取的，所以你需要定期的清理一下。当然，重要数据不可放置在此目录啊！ 因为FHS甚至建议在开机时，应该要将/tmp下的数据都删除唷！
/usr	第二层 FHS 设置，后续介绍
/var	第二曾 FHS 设置，主要为放置变动性的数据，后续介绍
第二部份：FHS建议可以存在的目录	
/home	这是系统默认的使用者主文件夹（home directory）。在你新增一个一般使用者帐号时， 默认的使用者主文件夹都会规范到这里来。比较重要的是，主文件夹有两种代号喔：~：代表目前这个使用者的主文件夹 ~dmtsai ：则代表 dmtsai 的主文件夹！
/lib<qual>	用来存放与 /lib 不同的格式的二进制函数库，例如支持 64 位的 /lib64 函数库等
/root	系统管理员（root）的主文件夹。之所以放在这里，是因为如果进入单人维护模式而仅挂载根目录时， 该目录就能够拥有root的主文件夹，所以我们会希望root的主文件夹与根目录放置在同一个分区中。

事实上FHS针对根目录所定义的标准就仅有上面的咚咚，不过我们的Linux下面还有许多目录你也需要了解一下的。 下面是几个在Linux当中也是非常重要的目录喔：

目录	应放置文件内容
/lost+found	这个目录是使用标准的ext2/ext3/ext4文件系统格式才会产生的一个目录，目的在于当文件系统发生错误时， 将一些遗失的片段放置到这个目录下。不过如果使用的是 xfs 文件系统的话，就不会存在这个目录了！
/proc	这个目录本身是一个“虚拟文件系统（virtual filesystem）”喔！他放置的数据都是在内存当中， 例如系统核心、行程信息（process）、周边设备的状态及网络状态等等。因为这个目录下的数据都是在内存当中， 所以本身不占任何硬盘空间啊！比较重要的文件例如：/proc/cpuinfo, /proc/dma, /proc/interrupts, /proc/ioports, /proc/net/* 等等。
/sys	这个目录其实跟/proc非常类似，也是一个虚拟的文件系统，主要也是记录核心与系统硬件信息较相关的信息。 包括目前已载入的核心模块与核心侦测到的硬件设备信息等等。这个目录同样不占硬盘容量喔！

早期 Linux 在设计的时候，若发生问题时，救援模式通常仅挂载根目录而已，因此有五个重要的目录被要求一定要与根目录放置在一起， 那就是 /etc, /bin, /dev, /lib, /sbin 这五个重要目录。现在许多的 Linux distributions 由于已经将许多非必要的文件移出 /usr 之外了， 所以 /usr 也是越来越精简，同时因为 /usr 被建议为“即使挂载成为只读，系统还是可以正常运行”的模样，所以救援模式也能同时挂载 /usr 喔！ 例如我们的这个 CentOS 7.x 版本在救援模式的情况下就是这样。因此那个五大目录的限制已经被打破了呦！例如 CentOS 7.x 就已经将 /sbin, /bin, /lib 通通移动到 /usr 下面了哩！

好了，谈完了根目录，接下来我们就来谈谈/usr以及/var啰！先看/usr里面有些什么东西：

- /usr 的意义与内容：

依据FHS的基本定义，/usr里面放置的数据属于可分享的与不可变动的（shareable, static）， 如果你知道如何通过网络进行分区的挂载（例如在服务器篇会谈到的NFS服务器），那么/usr确实可以分享给区域网络内的其他主机来使用喔！

很多读者都会误会/usr为user的缩写，其实usr是Unix Software Resource的缩写， 也就是“Unix操作系统软件资源”所放置的目录，而不是使用者的数据啦！这点要注意。 FHS建议所有软件开发者，应该将他们的数据合理的分别放置到这个目录下的次目录，而不要自行创建该软件自己独立的目录。

因为是所有系统默认的软件（distribution发布者提供的软件）都会放置到/usr下面，因此这个目录有点类似Windows 系统的“C:\Windows\ （当中的一部份） + C:\Program files\”这两个目录的综合体，系统刚安装完毕时，这个目录会占用最多的硬盘容量。一般来说，/usr的次目录建议有下面这些：

目录	应放置文件内容
第一部份： FHS 要求必须要存在的目录	
/usr/bin/	所有一般用户能够使用的指令都放在这里！目前新的 CentOS 7 已经将全部的使用者指令放置于此，而使用链接文件的方式将 /bin 链接至此！也就是说， /usr/bin 与 /bin 是一模一样了！另外，FHS 要求在此目录下不应该有子目录！
/usr/lib/	基本上，与 /lib 功能相同，所以 /lib 就是链接到此目录中的！
/usr/local/	系统管理员在本机自行安装自己下载的软件（非distribution默认提供者），建议安装到此目录， 这样会比较便于管理。举例来说，你的distribution提供的软件较旧，你想安装较新的软件但又不想移除旧版，此时你可以将新版软件安装于/usr/local/目录下，可与原先的旧版软件有分别啦！ 你可以自行到/usr/local去看看，该目录下也是具有bin, etc, include, lib...的次目录喔！
/usr/sbin/	非系统正常运行所需要的系统指令。最常见的就是某些网络服务器软件的服务指令（daemon）啰！不过基本功能与 /sbin 也差不多， 因此目前 /sbin 就是链接到此目录中的。
/usr/share/	主要放置只读架构的数据文件，当然也包括共享文件。在这个目录下放置的数据几乎是不分硬件架构均可读取的数据， 因为几乎都是文字文件嘛！在此目录下常见的还有这些次目录：/usr/share/man：线上说明文档 /usr/share/doc：软件杂项的文件说明 /usr/share/zoneinfo：与时区有关的时区文件
第二部份： FHS 建议可以存在的目录	
/usr/games/	与游戏比较相关的数据放置处
/usr/include/	c/c++等程序语言的文件开始（header）与包含档（include）放置处，当我们以tarball方式 （*.tar.gz 的方式安装软件）安装某些数据时，会使用到里头的许多包含档喔！
/usr/libexec/	某些不被一般使用者惯用的可执行文件或脚本（script）等等，都会放置在此目录中。例如大部分的 X 窗口下面的操作指令， 很多都是放在此目录下的。
/usr/lib<qual>/	与 /lib<qual>/功能相同，因此目前 /lib<qual> 就是链接到此目录中
/usr/src/	一般源代码建议放置到这里，src有source的意思。至于核心源代码则建议放置到/usr/src/linux/目录下。

- /var 的意义与内容：

如果/usr是安装时会占用较大硬盘容量的目录，那么/var就是在系统运行后才会渐渐占用硬盘容量的目录。 因为/var目录主要针对常态性变动的文件，包括高速缓存（cache）、登录文件（log file）以及某些软件运行所产生的文件， 包括程序文件（lock file, run file），或者例如MySQL数据库的文件等等。常见的次目录有：

目录	应放置文件内容
第一部份：FHS要求必须要存在的目录	
/var/cache/	应用程序本身运行过程中会产生的一些暂存盘；
/var/lib/	程序本身执行的过程中，需要使用到的数据文件放置的目录。在此目录下各自的软件应该要有各自的目录。 举例来说，MySQL的数据库放置到/var/lib/mysql/而rpm的数据库则放到/var/lib/rpm去！
/var/lock/	某些设备或者是文件资源一次只能被一个应用程序所使用，如果同时有两个程序使用该设备时， 就可能产生一些错误的状况，因此就得要将该设备上锁（lock），以确保该设备只会给单一软件所使用。 举例来说，烧录机正在烧录一块光盘，你想一下，会不会有两个人同时在使用一个烧录机烧片？ 如果两个人同时烧录，那片子写入的是谁的数据？所以当第一个人在烧录时该烧录机就会被上锁， 第二个人就得要该设备被解除锁定（就是前一个人用完了）才能够继续使用啰。目前此目录也已经挪到 /run/lock 中！
/var/log/	重要到不行！这是登录文件放置的目录！里面比较重要的文件如/var/log/messages, /var/log/wtmp（记录登陆者的信息）等。
/var/mail/	放置个人电子邮件信箱的目录，不过这个目录也被放置到/var/spool/mail/目录中！ 通常这两个目录是互为链接文件啦！
/var/run/	某些程序或者是服务启动后，会将他们的PID放置在这个目录下喔！至于PID的意义我们会在后续章节提到的。 与 /run 相同，这个目录链接到 /run 去了！
/var/spool/	这个目录通常放置一些伫列数据，所谓的“伫列”就是排队等待其他程序使用的数据啦！ 这些数据被使用后通常都会被删除。举例来说，系统收到新信会放置到/var/spool/mail/中， 但使用者收下该信件后该封信原则上就会被删除。信件如果暂时寄不出去会被放到/var/spool/mqueue/中， 等到被送出后就被删除。如果是工作调度数据（crontab），就会被放置到/var/spool/cron/目录中！

建议在你读完整个基础篇之后，可以挑战FHS官方英文文件（参考本章参考数据），相信会让你对于Linux操作系统的目录有更深入的了解喔！

- 针对FHS，各家distributions的异同，与 CentOS7 的变化

由于FHS仅是定义出最上层（/）及次层（/usr, /var）的目录内容应该要放置的文件或目录数据， 因此，在其他次目录层级内，就可以随开发者自行来配置了。举例来说，CentOS的网络设置数据放在 /etc/sysconfig/network-scripts/ 目录下，但是SuSE则是将网络放置在 /etc/sysconfig/network/ 目录下，目录名称可是不同的呢！不过只要记住大致的FHS标准，差异性其实有限啦！

此外，CentOS 7 在目录的编排上与过去的版本不同喔！本节稍早之前已经有介绍过，这里做个汇整。 比较大的差异在于将许多原本应该要在根目录（/）里面的目录，将他内部数据全部挪到 /usr 里面去，然后进行链接设置！包括下面这些：

- /bin --> /usr/bin
- /sbin --> /usr/sbin
- /lib --> /usr/lib
- /lib64 --> /usr/lib64
- /var/lock --> /run/lock
- /var/run --> /run

5.3.2 目录树（directory tree）

另外，在Linux下面，所有的文件与目录都是由根目录开始的！那是所有目录与文件的源头～然后再一个一个的分支下来，有点像是树枝状啊～因此，我们也称这种目录配置方式为："目录树（directory tree）" 这个目录树有什么特性呢？他主要的特性有：

- 目录树的启始点为根目录 （/, root）；
- 每一个目录不止能使用本地端的 partition 的文件系统，也可以使用网络上的 filesystem 。举例来说， 可以利用 Network File System （NFS） 服务器挂载某特定目录等。
- 每一个文件在此目录树中的文件名（包含完整路径）都是独一无二的。

好，谈完了FHS的标准之后，实际来看看CentOS在根目录下面会有什么样子的数据吧！我们可以下达以下的指令来查询：

```
[dmtsai@study ~]$ ls -l /
lrwxrwxrwx.   1 root root    7 May  4 17:51 bin -&gt; usr/bin
dr-xr-xr-x.   4 root root 4096 May  4 17:59 boot
drwxr-xr-x.  20 root root 3260 Jun  2 19:27 dev
drwxr-xr-x. 131 root root 8192 Jun  2 23:51 etc
drwxr-xr-x.   3 root root   19 May  4 17:56 home
lrwxrwxrwx.   1 root root    7 May  4 17:51 lib -&gt; usr/lib
lrwxrwxrwx.   1 root root    9 May  4 17:51 lib64 -&gt; usr/lib64
drwxr-xr-x.   2 root root    6 Jun 10  2014 media
drwxr-xr-x.   2 root root    6 Jun 10  2014 mnt
drwxr-xr-x.   3 root root   15 May  4 17:54 opt
dr-xr-xr-x. 154 root root    0 Jun  2 11:27 proc
dr-xr-x---.   5 root root 4096 Jun  3 00:04 root
drwxr-xr-x.  33 root root  960 Jun  2 19:27 run
lrwxrwxrwx.   1 root root    8 May  4 17:51 sbin -&gt; usr/sbin
drwxr-xr-x.   2 root root    6 Jun 10  2014 srv
dr-xr-xr-x.  13 root root    0 Jun  2 19:27 sys
drwxrwxrwt.  12 root root 4096 Jun  3 19:48 tmp
drwxr-xr-x.  13 root root 4096 May  4 17:51 usr
drwxr-xr-x.  22 root root 4096 Jun  2 19:27 var
```

上述目录相关的介绍都在上一个小节，要记得回去查看看。如果我们将整个目录树以图示的方法来显示，并且将较为重要的文件数据列出来的话，那么目录树架构有点像这样：

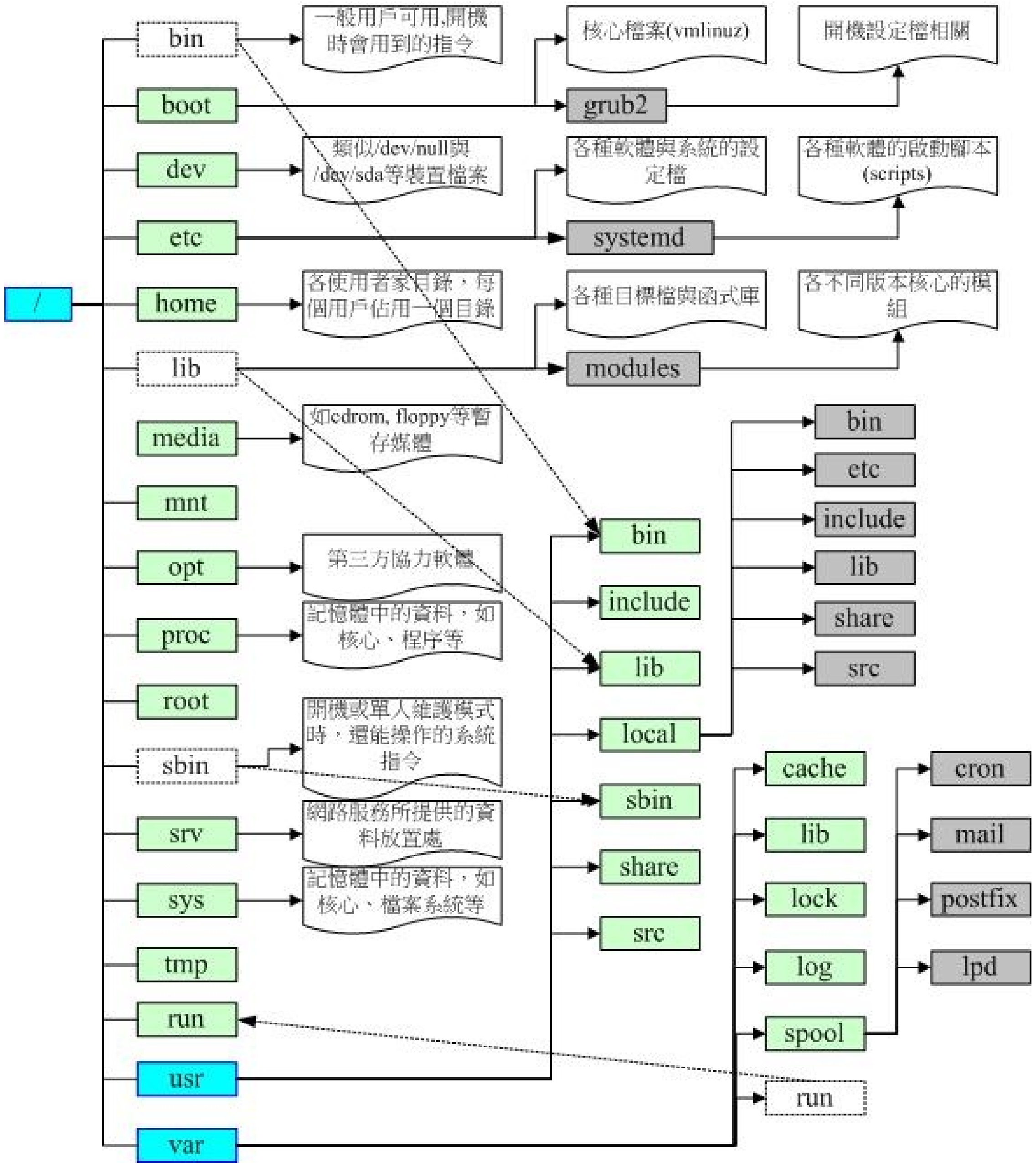


图5.3.1、目录树架构示意图

鸟哥只有就各目录进行简单的解释，看看就好，详细的解释请回到刚刚说明的表格中去查阅喔！ 看完了FHS标准之后，现在回到第二章里面去看看安装前Linux规划的分区情况， 对于当初为何需要分区为这样的情况，有点想法了吗？^_^。根据FHS的定义，你最好能够将/var独立出来， 这样对于系统的数据还有一些安全性的保护呢！因为至少/var死掉时，你的根目录还会活着嘛！ 还能够进入救援模式啊！

5.3.3 绝对路径与相对路径

除了需要特别注意的FHS目录配置外，在文件名部分我们也要特别注意喔！因为根据文件名写法的不同，也可将所谓的路径（path）定义为绝对路径（absolute）与相对路径（relative）。 这两种文件名/路径的写法依据是这样的：

- 绝对路径：由根目录（/）开始写起的文件名或目录名称， 例如 /home/dmtsai/.bashrc；
- 相对路径：相对于目前路径的文件名写法。 例如 ./home/dmtsai 或 ../../home/dmtsai/ 等等。反正开头不是 / 就属于相对路径的写法

而你必须要了解，相对路径是以“你当前所在路径的相对位置”来表示的。举例来说，你目前在 /home 这个目录下， 如果想要进入 /var/log 这个目录时，可以怎么写呢？

1. cd /var/log （absolute）
2. cd ../var/log （relative）

因为你在 /home 下面，所以要回到上一层 （../） 之后，才能继续往 /var 来移动的！ 特别注意这两个特殊的目录：

- . ：代表当前的目录，也可以使用 ./ 来表示；
- .. ：代表上一层目录，也可以 ../ 来代表。

这个 . 与 .. 目录概念是很重要的，你常常会看到 cd .. 或 ./command 之类的指令下达方式，就是代表上一层与目前所在目录的工作状态喔！很重要的呐！

例题：如何先进入/var/spool/mail/目录，再进入到/var/spool/cron/目录内？答：由于/var/spool/mail与/var/spool/cron是同样在/var/spool/目录中，因此最简单的指令下达方法为：

1. cd /var/spool/mail
2. cd ../cron

如此就不需要在由根目录开始写起了。这个相对路径是非常有帮助的！尤其对于某些软件开发商来说。 一般来说，软件开发商会将数据放置到/usr/local/里面的各相对目录，你可以参考图3.2.1的相对位置。 但如果使用者想要安装到不同目录呢？就得要使用相对路径啰！^_^

例题：网络文件常常提到类似“./run.sh”之类的数据，这个指令的意义为何？答：由于指令的执行需要变量（bash章节才会提到）的支持，若你的可执行文件放置在本目录，并且本目录并非正规的可执行文件目录（/bin, /usr/bin等为正规），此时要执行指令就得要严格指定该可执行文件。“./”代表“本目录”的意思，所以“./run.sh”代表“执行本目录下， 名为run.sh的文件”啰！

5.3.4 CentOS 的观察

如同在第一章谈到的 Linux distribution 的差异性， 除了 FHS 之外，还有个 Linux Standard Base （LSB） 的标准是可以依循的！我们可以简单的使用 ls 来查看 FHS 规范的目录是否正确的存在于你的 Linux 系统中， 那么 Linux 核心、LSB 的标准又该如何查阅呢？基本上，LSB 团队是有列出正确支持 LSB 标准的 distribution 在如下的网页中：

- https://www.linuxbase.org/lsb-cert/productdir.php?by_lsb

不过，如果你想要知道确切的核心与 LSB 所需求的几种重要的标准的话，恐怕就得要使用诸如 uname 与 lsb_release 等指令来查阅了。 不过，这个 lsb_release 指令已经不是默认安装的软件了，所以你得要自己安装该软件才才行。因为我们尚未讲到网络与挂载等动作， 所以下面的安装流程在你的机器上面应该是无法执行的 （除非你确实可以连上 Internet 才行！），因为 CentOS7 在这个软件上面实在有太多的相依软件， 所以无法单纯使用 rpm 来安装！若你有公开的网络，那么下面的指令才能够顺利运行！

```
# 1\. 通过 uname 检查 Linux 核心与操作系统的位版本
[dmtsai@study ~]$ uname -r   # 查看核心版本
3.10.0-229.el7.x86_64
[dmtsai@study ~]$ uname -m   # 查看操作系统的位版本
x86_64

# 2\. 假设你的 CentOS 7 确实有网络可以使用的情况下 （要用 root 的身份）
[root@study ~]# yum install redhat-lsb   # yum 的用法后面章节才会介绍
.....（前面省略）....
Install  1 Package   （+85 Dependent packages）
Upgrade              （  4 Dependent packages）

Total size: 47 M
Total download size: 31 M
Is this ok [y/d/N]: y
.....（后面省略）....
Retrieving key from file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
Importing GPG key 0xF4A80EB5:
 Userid     : "CentOS-7 Key （CentOS 7 Official Signing Key） &lt;security@centos.org&gt;'
 Fingerprint: 6341 ab27 53d7 8a78 a7c2 7bb1 24c6 a8a7 f4a8 0eb5
 Package    : centos-release-7-0.1406.el7.centos.2.3.x86_64 （@anaconda）
 From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
Is this ok [y/N]: y
.....（后面省略）....

[root@study ~]# lsb_release -a
LSB Version:    :core-4.1-amd64:core-4.1-noarch:cxx-4.1-amd64:cxx-4.1-noarch:
desktop-4.1-amd64:desktop-4.1-noarch:languages-4.1-amd64:languages-4.1-noarch:
printing-4.1-amd64:printing-4.1-noarch  # LSB 的相关版本
Distributor ID: CentOS
Description:    CentOS Linux release 7.0.1406 （Core）
Release:        7.0.1406
Codename:       Core
```

这个 lsb*release* 的东西大家先看看就好，因为有牵涉到后面的 *yum* 软件安装的东西，这部份我们还没有谈到啊～而且如果你现在就直接安装， 未来我们谈网络与软件的阶段时，恐怕有些地方会跟我们的测试机环境不同～所以...先看看就好喔！ ^^



Tips 在这里要跟大家说抱歉，因为不想要破坏整体测试机器的环境，所以鸟哥使用了另一部虚拟机来安装 redhat-lsb 这套软件，而另一部虚拟机是通过 CentOS 7.0 而非 CentOS 7.1 的版本，因此你应该会发现到上面使用 lsb_release 指令的输出中，竟然出现了 7.0.1406 的东东～眞是不好意思～

5.4 重点回顾

- Linux的每个文件中，可分别给予使用者、群组与其他人三种身份个别的 rwx 权限；
- 群组最有用的功能之一，就是当你在团队开发资源的时候，且每个帐号都可以有多个群组的支持；
- 利用ls -l显示的文件属性中，第一个字段是文件的权限，共有十个位，第一个位是文件类型， 接下来三个为一组共三组，为使用者、群组、其他人的权限，权限有r,w,x三种；
- 如果文件名之前多一个“ . ”，则代表这个文件为“隐藏文件”；
- 若需要root的权限时，可以使用 su - 这个指令来切换身份。处理完毕则使用 exit 离开 su 的指令环境。
- 更改文件的群组支持可用chgrp，修改文件的拥有者可用chown，修改文件的权限可用chmod
- chmod修改权限的方法有两种，分别是符号法与数字法，数字法中r,w,x分数为4,2,1；
- 对文件来讲，权限的性能为：
    - r：可读取此一文件的实际内容，如读取文本文件的文字内容等；
    - w：可以编辑、新增或者是修改该文件的内容（但不含删除该文件）；
    - x：该文件具有可以被系统执行的权限。
- 对目录来说，权限的性能为：
    - r （read contents in directory）
    - w （modify contents of directory）
    - x （access directory）
- 要开放目录给任何人浏览时，应该至少也要给予r及x的权限，但w权限不可随便给；
- 能否读取到某个文件内容，跟该文件所在的目录权限也有关系 （目录至少需要有 x 的权限）。
- Linux文件名的限制为：单一文件或目录的最大容许文件名为 255 个英文字符或 128 个中文字符；
- 根据FHS的官方文件指出， 他们的主要目的是希望让使用者可以了解到已安装软件通常放置于那个目录下
- FHS订定出来的四种目录特色为：shareable, unshareable, static, variable等四类；
- FHS所定义的三层主目录为：/, /var, /usr三层而已；
- 绝对路径文件名为从根目录 / 开始写起，否则都是相对路径的文件名。

5.5 本章练习

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ）

- 早期的 Unix 系统文件名最多允许 14 个字符，而新的 Unix 与 Linux 系统中，文件名最多可以容许几个字符？由于使用Ext2/Ext3/Ext4/xfs 文件系统，单一文件名可达 255 字符
- 当一个一般文件权限为 -rwxrwxrwx 则表示这个文件的意义为？任何人皆可读取、修改或编辑、可以执行，但不一定能删除。
- 我需要将一个文件的权限改为 -rwxr-xr-- 请问该如何下达指令？chmod 754 filename 或 chmod u=rwx,g=rx,o=r filename
- 若我需要更改一个文件的拥有者与群组，该用什么指令？chown, chgrp
- 请问下面的目录与主要放置什么数据： /etc/, /boot, /usr/bin, /bin, /usr/sbin, /sbin, /dev, /var/log, /run
    - /etc/：几乎系统的所有设置文件均在此，尤其 passwd,shadow
    - /boot：开机配置文件，也是默认摆放核心 vmlinuz 的地方
    - /usr/bin, /bin：一般可执行文件摆放的地方
    - /usr/sbin, /sbin：系统管理员常用指令集
    - /dev：摆放所有系统设备文件的目录
    - /var/log：摆放系统登录文件的地方
    - /run：CentOS 7 以后才有，将经常变动的项目（每次开机都不同，如程序的PID）移动到内存暂存，所以 /run 并不占实际磁盘容量
- 若一个文件的文件名开头为“ . ”，例如 .bashrc 这个文件，代表什么？另外，如何显示出这个文件名与他的相关属性？有“ . ”为开头的为隐藏文件，需要使用 ls -a 这个 -a 的选项才能显示出隐藏文件的内容，而使用 ls -al 才能显示出属性。

5.6 参考资料与延伸阅读

- [1]各种文件系统的文件名长度限制，维基百科： http://en.wikipedia.org/wiki/Comparison_of_file_systems
- [2]FHS 标准的相关说明： 维基百科简易说明： http://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard FHS 2.3 （2004 年版）的标准文件：http://www.pathname.com/fhs/pub/fhs-2.3.html FHS 3.0 （2015 年版）的标准文件：http://refspecs.linuxfoundation.org/FHS_3.0/fhs-3.0.pdf
- 关于 Journaling 日志式文章的相关说明 http://www.linuxplanet.com/linuxplanet/reports/3726/1/

2002/07/18：第一次完成 2003/02/06：重新编排与加入FAQ 2005/06/28：将旧的数据移动到这里 2005/07/15：呼呼～终于改完成了～这次的修订当中，加入了 FHS 的说明，希望大家能够比较清楚 Linux 的目录配置！ 2005/08/05：修订了最大文件名字符，应该是 255 才对！另外，加入了“文件名限制”的部分！ 2005/09/03：修订了目录权限相关的说明，将原本仅具有 r 却写成无法使用 ls 浏览的说明数据移除！ 2008/09/08：旧的针对FC4所写的文章移动到此处 2008/09/20：针对FHS加强说明了一下，分为/, /usr, /var三层来个别说明！并非抄袭官网的数据而已喔！ 2008/09/23：经过一场大感冒，停工了四、五天，终于还是给他完工了！^_^ 2008/10/21：原本的第四小节 Linux 的文件系统，因为与第八章重复性太高，将他移除了！ 2009/08/01：加入了 lsb_release 的相关说明！ 2009/08/18：调整一下显示的情况，使得更易读～ 2015/06/02：将原本基于 CentOS 5.x 撰写的就文章放在这里了喔！

第六章、Linux 文件与目录管理

最近更新日期：20//

在前一章我们认识了Linux系统下的文件权限概念以及目录的配置说明。 在这个章节当中，我们就直接来进一步的操作与管理文件及目录吧！包括在不同的目录间变换、 创建与删除目录、创建与删除文件，还有寻找文件、查阅文件内容等等，都会在这个章节作个简单的介绍啊！

6.1 目录与路径

由前一章Linux的文件权限与目录配置中通过FHS了解了Linux的“树状目录”概念之后， 接下来就得要实际的来搞定一些基本的路径问题了！这些目录的问题当中，最重要的莫过于前一章也谈过的“绝对路径”与“相对路径”的意义啦！ 绝对/相对路径的写法并不相同，要特别注意。此外，当你下达指令时，该指令是通过什么功能来取得的？ 这与PATH这个变量有关呢！下面就让我们来谈谈啰！

6.1.1 相对路径与绝对路径

在开始目录的切换之前，你必须要先了解一下所谓的“路径（PATH）”， 有趣的是：什么是“相对路径”与“绝对路径”？ 虽然前一章已经稍微针对这个议题提过一次，不过，这里不厌其烦的再次的强调一下！

- 绝对路径：路径的写法“一定由根目录 / 写起”，例如： /usr/share/doc 这个目录。
- 相对路径：路径的写法“不是由 / 写起”，例如由 /usr/share/doc 要到 /usr/share/man 下面时，可以写成： “cd ../man”这就是相对路径的写法啦！相对路径意指“相对于目前工作目录的路径！”

- 相对路径的用途

那么相对路径与绝对路径有什么了不起呀？喝！那可真的是了不起了！假设你写了一个软件， 这个软件共需要三个目录，分别是 etc, bin, man 这三个目录，然而由于不同的人喜欢安装在不同的目录之下， 假设甲安装的目录是 /usr/local/packages/etc, /usr/local/packages/bin 及 /usr/local/packages/man ，不过乙却喜欢安装在 /home/packages/etc, /home/packages/bin, /home/packages/man 这三个目录中，请问如果需要用到绝对路径的话，那么是否很麻烦呢？是的！ 如此一来每个目录下的东西就很难对应的起来！这个时候相对路径的写法就显的特别的重要了！

此外，如果你跟鸟哥一样，喜欢将路径的名字写的很长，好让自己知道那个目录是在干什么的，例如： /cluster/raid/output/taiwan2006/smoke 这个目录，而另一个目录在 /cluster/raid/output/taiwan2006/cctm ，那么我从第一个要到第二个目录去的话，怎么写比较方便？ 当然是“ cd ../cctm ”比较方便啰！对吧！

- 绝对路径的用途

但是对于文件名的正确性来说，“绝对路径的正确度要比较好～”。 一般来说，鸟哥会建议你，如果是在写程序 （shell scripts） 来管理系统的条件下，务必使用绝对路径的写法。 怎么说呢？因为绝对路径的写法虽然比较麻烦，但是可以肯定这个写法绝对不会有问题。 如果使用相对路径在程序当中，则可能由于你执行的工作环境不同，导致一些问题的发生。 这个问题在工作调度（at, cron, 第十五章）当中尤其重要！这个现象我们在十二章、shell script 时，会再次的提醒你喔！ ^_^

6.1.2 目录的相关操作

我们之前稍微提到变换目录的指令是cd，还有哪些可以进行目录操作的指令呢？ 例如创建目录啊、删除目录之类的～还有，得要先知道的，就是有哪些比较特殊的目录呢？ 举例来说，下面这些就是比较特殊的目录，得要用力的记下来才行：

```
.          代表此层目录
..         代表上一层目录
-          代表前一个工作目录
~          代表“目前使用者身份”所在的主文件夹
~account   代表 account 这个使用者的主文件夹（account是个帐号名称）
```

需要特别注意的是：在所有目录下面都会存在的两个目录，分别是“.”与“..” 分别代表此层与上层目录的意思。那么来思考一下下面这个例题：

例题：请问在Linux下面，根目录下有没有上层目录（..）存在？答：若使用“ ls -al / ”去查询，可以看到根目录下确实存在 . 与 .. 两个目录，再仔细的查阅， 可发现这两个目录的属性与权限完全一致，这代表根目录的上一层（..）与根目录自己（.）是同一个目录。

下面我们就来谈一谈几个常见的处理目录的指令吧：

- cd：变换目录
- pwd：显示目前的目录
- mkdir：创建一个新的目录
- rmdir：删除一个空的目录

- cd （change directory, 变换目录）

我们知道dmtsai这个使用者的主文件夹是/home/dmtsai/，而root主文件夹则是/root/，假设我以root身份在 Linux系统中，那么简单的说明一下这几个特殊的目录的意义是：

```
[dmtsai@study ~]$ su -  # 先切换身份成为 root 看看！
[root@study ~]# cd [相对路径或绝对路径]
# 最重要的就是目录的绝对路径与相对路径，还有一些特殊目录的符号啰！
[root@study ~]# cd ~dmtsai
# 代表去到 dmtsai 这个使用者的主文件夹，亦即 /home/dmtsai
[root@study dmtsai]# cd ~
# 表示回到自己的主文件夹，亦即是 /root 这个目录
[root@study ~]# cd
# 没有加上任何路径，也还是代表回到自己主文件夹的意思喔！
[root@study ~]# cd ..
# 表示去到目前的上层目录，亦即是 /root 的上层目录的意思；
[root@study /]# cd -
# 表示回到刚刚的那个目录，也就是 /root 啰～
[root@study ~]# cd /var/spool/mail
# 这个就是绝对路径的写法！直接指定要去的完整路径名称！
[root@study mail]# cd ../postfix
# 这个是相对路径的写法，我们由/var/spool/mail 去到/var/spool/postfix 就这样写！
```

cd是Change Directory的缩写，这是用来变换工作目录的指令。注意，目录名称与cd指令之间存在一个空格。 一登陆Linux系统后，每个帐号都会在自己帐号的主文件夹中。那回到上一层目录可以用“ cd .. ”。 利用相对路径的写法必须要确认你目前的路径才能正确的去到想要去的

目录。例如上表当中最后一个例子， 你必须要确认你是在/var/spool/mail当中，并且知道在/var/spool当中有个mqueue的目录才行啊～ 这样才能使用cd ../postfix 去到正确的目录说，否则就要直接输入cd /var/spool/postfix 啰～

其实，我们的提示字符，亦即那个 [root@study ~]# 当中，就已经有指出目前的目录了， 刚登陆时会到自己的主文件夹，而主文件夹还有一个代码，那就是“ ~ ”符号！ 例如上面的例子可以发现，使用“ cd ~ ”可以回到个人的主文件夹里头去呢！ 另外，针对 cd 的使用方法，如果仅输入 cd 时，代表的就是“ cd ~ ”的意思喔～ 亦即是会回到自己的主文件夹啦！而那个“ cd - ”比较难以理解，请自行多做几次练习，就会比较明白了。



Tips 还是要一再地提醒，我们的 Linux 的默认命令行界面 （bash shell） 具有文件补齐功能， 你要常常利用 [tab] 按键来达成你的目录完整性啊！这可是个好习惯啊～可以避免你按错键盘输入错字说～ ^_^

- pwd （显示目前所在的目录）

```
[root@study ~]# pwd [-P]
选项与参数：
-P  ：显示出确实的路径，而非使用链接 （link） 路径。

范例：单纯显示出目前的工作目录：
[root@study ~]# pwd
/root   &lt;== 显示出目录啦～

范例：显示出实际的工作目录，而非链接文件本身的目录名而已
[root@study ~]# cd /var/mail   &lt;==注意，/var/mail是一个链接文件
[root@study mail]# pwd
/var/mail         &lt;==列出目前的工作目录
[root@study mail]# pwd -P
/var/spool/mail   &lt;==怎么回事？有没有加 -P 差很多～
[root@study mail]# ls -ld /var/mail
lrwxrwxrwx. 1 root root 10 May  4 17:51 /var/mail -&gt; spool/mail
# 看到这里应该知道为啥了吧？因为 /var/mail 是链接文件，链接到 /var/spool/mail
# 所以，加上 pwd -P 的选项后，会不以链接文件的数据显示，而是显示正确的完整路径啊！
```

pwd是Print Working Directory的缩写，也就是显示目前所在目录的指令， 例如在上个表格最后的目录是/var/mail这个目录，但是提示字符仅显示mail， 如果你想要知道目前所在的目录，可以输入pwd即可。此外，由于很多的套件所使用的目录名称都相同，例如 /usr/local/etc还有/etc，但是通常Linux仅列出最后面那一个目录而已，这个时候你就可以使用pwd 来知道你的所在目录啰！免得搞错目录，结果...

其实有趣的是那个 -P 的选项啦！他可以让我们取得正确的目录名称，而不是以链接文件的路径来显示的。 如果你使用的是CentOS 7.x的话，刚刚好/var/mail是/var/spool/mail的链接文件， 所以，通过到/var/mail下达pwd -P就能够知道这个选项的意义啰～ ^_^

- mkdir （创建新目录）

```
[root@study ~]# mkdir [-mp] 目录名称
选项与参数：
-m ：设置文件的权限喔！直接设置，不需要看默认权限 （umask） 的脸色～
-p ：帮助你直接将所需要的目录（包含上层目录）递回创建起来！

范例：请到/tmp下面尝试创建数个新目录看看：
[root@study ~]# cd /tmp
[root@study tmp]# mkdir test    &lt;==创建一名为 test 的新目录
[root@study tmp]# mkdir test1/test2/test3/test4
mkdir: cannot create directory ‘test1/test2/test3/test4’: No such file or directory
# 话说，系统告诉我们，没可能创建这个目录啊！就是没有目录才要创建的！见鬼嘛？
[root@study tmp]# mkdir -p test1/test2/test3/test4
# 原来是要建 test4 上层没先建 test3 之故！加了这个 -p 的选项，可以自行帮你创建多层目录！

范例：创建权限为rwx--x--x的目录
[root@study tmp]# mkdir -m 711 test2
[root@study tmp]# ls -ld test*
drwxr-xr-x. 2 root   root  6 Jun  4 19:03 test
drwxr-xr-x. 3 root   root 18 Jun  4 19:04 test1
drwx--x--x. 2 root   root  6 Jun  4 19:05 test2
# 仔细看上面的权限部分，如果没有加上 -m 来强制设置属性，系统会使用默认属性。
# 那么你的默认属性为何？这要通过下面介绍的 [umask](../Text/index.html#umask) 才能了解喔！ ^_^
```

如果想要创建新的目录的话，那么就使用mkdir （make directory）吧！ 不过，在默认的情况下， 你所需要的目录得一层一层的创建才行！例如：假如你要创建一个目录为 /home/bird/testing/test1，那么首先必须要有 /home 然后 /home/bird ，再来 /home/bird/testing 都必须要存在，才可以创建 /home/bird/testing/test1 这个目录！假如没有 /home/bird/testing 时，就没有办法创建 test1 的目录啰！

不过，现在有个更简单有效的方法啦！那就是加上 -p 这个选项喔！你可以直接下达：“ mkdir -p /home/bird/testing/test1 ” 则系统会自动的帮你将 /home, /home/bird, /home/bird/testing 依序的创建起目录！并且， 如果该目录本来就已经存在时，系统也不会显示错误讯息喔！挺快乐的吧！ ^_^。 不过鸟哥不建议常用-p这个选项，因为担心如果你打错字，那么目录名称就会变的乱七八糟的！

另外，有个地方你必须要先有概念，那就是“默认权限”的地方。我们可以利用 -m 来强制给予一个新的目录相关的权限， 例如上表当中，我们给予 -m 711 来给予新的目录 drwx--x--x 的权限。不过，如果没有给予 -m 选项时， 那么默认的新建目录权限又是什么呢？这个跟 umask 有关，我们在本章后头会加以介绍的。

- rmdir （删除“空”的目录）

```
[root@study ~]# rmdir [-p] 目录名称
选项与参数：
-p ：连同“上层”“空的”目录也一起删除

范例：将于mkdir范例中创建的目录（/tmp下面）删除掉！
[root@study tmp]# ls -ld test*   &lt;==看看有多少目录存在？
drwxr-xr-x. 2 root   root  6 Jun  4 19:03 test
drwxr-xr-x. 3 root   root 18 Jun  4 19:04 test1
drwx--x--x. 2 root   root  6 Jun  4 19:05 test2
[root@study tmp]# rmdir test   &lt;==可直接删除掉，没问题
[root@study tmp]# rmdir test1  &lt;==因为尚有内容，所以无法删除！
rmdir: failed to remove ‘test1’: Directory not empty
[root@study tmp]# rmdir -p test1/test2/test3/test4
[root@study tmp]# ls -ld test*    &lt;==您看看，下面的输出中test与test1不见了！
drwx--x--x. 2 root   root  6 Jun  4 19:05 test2
# 瞧！利用 -p 这个选项，立刻就可以将 test1/test2/test3/test4 一次删除～
# 不过要注意的是，这个 rmdir 仅能“删除空的目录”喔！
```

如果想要删除旧有的目录时，就使用rmdir吧！例如将刚刚创建的test杀掉，使用“ rmdir test ”即可！请注意哟！目录需要一层一层的删除才行！而且被删除的目录里面必定不能存在其他的目录或文件！ 这也是所谓的空的目录（empty directory）的意思啊！那如果要将所有目录下的东西都杀掉呢？！ 这个时候就必须使用“ rm -r test ”啰！不过，还是使用 rmdir 比较不危险！你也可以尝试以 -p 的选项加入，来删除上层的目录喔！

6.1.3 关于可执行文件路径的变量：$PATH

经过前一章FHS的说明后，我们知道查阅文件属性的指令ls完整文件名为：/bin/ls（这是绝对路径）， 那你会不会觉得很奇怪：“为什么我可以在任何地方执行/bin/ls这个指令呢？” 为什么我在任何目录下输入 ls 就一定可以显示出一些讯息而不会说找不到该 /bin/ls 指令呢？ 这是因为环境变量 PATH 的帮助所致呀！

当我们在执行一个指令的时候，举例来说“ls”好了，系统会依照PATH的设置去每个PATH定义的目录下搜寻文件名为ls的可可执行文件， 如果在PATH定义的目录中含有多个文件名为ls的可可执行文件，那么先搜寻到的同名指令先被执行！

现在，请下达“echo $PATH”来看看到底有哪些目录被定义出来了？ echo有“显示、印出”的意思，而 PATH 前面加的 $ 表示后面接的是变量，所以会显示出目前的 PATH ！

```
范例：先用root的身份列出搜寻的路径为何？
[root@study ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin

范例：用dmtsai的身份列出搜寻的路径为何？
[root@study ~]# exit    # 由之前的 su - 离开，变回原本的帐号！或再取得一个终端机皆可！
[dmtsai@study ~]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/bi
# 记不记得我们前一章说过，目前 /bin 是链接到 /usr/bin 当中的喔！
```

PATH（一定是大写）这个变量的内容是由一堆目录所组成的，每个目录中间用冒号（:）来隔开， 每个目录是有“顺序”之分的。仔细看一下上面的输出，你可以发现到无论是root还是dmtsai都有 /bin 或 /usr/bin 这个目录在PATH变量内，所以当然就能够在任何地方执行ls来找

到/bin/ls可执行文件啰！因为 /bin 在 CentOS 7 当中，就是链接到 /usr/bin 去的！ 所以这两个目录内容会一模一样！

我们用几个范例来让你了解一下，为什么PATH是那么重要的项目！

例题：假设你是root，如果你将ls由/bin/ls移动成为/root/ls（可用“mv /bin/ls /root”指令达成），然后你自己本身也在/root目录下， 请问（1）你能不能直接输入ls来执行？（2）若不能，你该如何执行ls这个指令？（3）若要直接输入ls即可执行，又该如何进行？答：由于这个例题的重点是将某个可执行文件移动到非正规目录去，所以我们先要进行下面的动作才行：（务必先使用 su - 切换成为root的身份）

```
[root@study ~]# mv /bin/ls /root
# mv 为移动，可将文件在不同的目录间进行移动作业
```

（1）接下来不论你在那个目录下面输入任何与ls相关的指令，都没有办法顺利的执行ls了！也就是说，你不能直接输入ls来执行，因为/root这个目录并不在PATH指定的目录中， 所以，即使你在/root目录下，也不能够搜寻到ls这个指令！

（2）因为这个ls确实存在于/root下面，并不是被删除了！所以我们可以通过使用绝对路径或者是相对路径直接指定这个可执行文件文件名， 下面的两个方法都能够执行ls这个指令：

```
[root@study ~]# /root/ls  &lt;==直接用绝对路径指定该文件名
[root@study ~]# ./ls      &lt;==因为在 /root 目录下，就用./ls来指定
```

（3）如果想要让root在任何目录均可执行/root下面的ls，那么就将/root加入PATH当中即可。 加入的方法很简单，就像下面这样：

```
[root@study ~]# PATH="${PATH}:/root"
```

上面这个作法就能够将/root加入到可执行文件搜寻路径PATH中了！不相信的话请您自行使用“echo $PATH”去查看吧！ 另外，除了 $PATH 之外，如果想要更明确的定义出变量的名称，可以使用大括号 ${PATH} 来处理变量的调用喔！ 如果确定这个例题进行没有问题了，请将ls搬回/bin下面，不然系统会挂点的！

```
[root@study ~]# mv /root/ls /bin
```

某些情况下，即使你已经将 ls 搬回 /bin 了，不过系统还是会告知你无法处理 /root/ls 喔！很可能是因为指令参数被高速缓存的关系。 不要紧张，只要登出 （exit） 再登陆 （su -） 就可以继续快乐的使用 ls 了！

例题：如果我有两个ls指令在不同的目录中，例如/usr/local/bin/ls与/bin/ls那么当我下达 ls 的时候，哪个ls会被执行？答：那还用说，就找出 ${PATH} 里面哪个目录先被查询，则那个目录下的指令就会被先执行了！所以用 dmtsai 帐号为例，他最先搜寻的是 /usr/local/bin， 所以

/usr/local/bin/ls 会先被执行喔！

例题：为什么 ${PATH} 搜寻的目录不加入本目录（.）？加入本目录的搜寻不是也不错？答：如果在PATH中加入本目录（.）后，确实我们就能够在指令所在目录进行指令的执行了。 但是由于你的工作目录并非固定（常常会使用cd来切换到不同的目录）， 因此能够执行的指令会有变动（因为每个目录下面的可可执行文件都不相同嘛！），这对使用者来说并非好事。

另外，如果有个坏心使用者在/tmp下面做了一个指令，因为/tmp是大家都能够写入的环境，所以他当然可以这样做。 假设该指令可能会窃取使用者的一些数据，如果你使用root的身份来执行这个指令，那不是很糟糕？ 如果这个指令的名称又是经常会被用到的ls时，那“中标”的概率就更高了！

所以，为了安全起见，不建议将“.”加入PATH的搜寻目录中。

而由上面的几个例题我们也可以知道几件事情：

- 不同身份使用者默认的PATH不同，默认能够随意执行的指令也不同（如root与dmtsai）；
- PATH是可以修改的；
- 使用绝对路径或相对路径直接指定某个指令的文件名来执行，会比搜寻PATH来的正确；
- 指令应该要放置到正确的目录下，执行才会比较方便；
- 本目录（.）最好不要放到PATH当中。

对于PATH更详细的“变量”说明，我们会在第三篇的bash shell中详细说明的！

6.2 文件与目录管理

谈了谈目录与路径之后，再来讨论一下关于文件的一些基本管理吧！文件与目录的管理上，不外乎“显示属性”、“拷贝”、“删除文件”及“移动文件或目录”等等，由于文件与目录的管理在 Linux 当中是很重要的， 尤其是每个人自己主文件夹的数据也都需要注意管理！所以我们来谈一谈有关文件与目录的一些基础管理部分吧！

6.2.1 文件与目录的检视： ls

```
[root@study ~]# ls [-aAdfFhilnrRSt] 文件名或目录名称..
[root@study ~]# ls [--color={never,auto,always}] 文件名或目录名称..
[root@study ~]# ls [--full-time] 文件名或目录名称..
选项与参数：
-a  ：全部的文件，连同隐藏文件（ 开头为 . 的文件） 一起列出来（常用）
-A  ：全部的文件，连同隐藏文件，但不包括 . 与 .. 这两个目录
-d  ：仅列出目录本身，而不是列出目录内的文件数据（常用）
-f  ：直接列出结果，而不进行排序 （ls 默认会以文件名排序！）
-F  ：根据文件、目录等信息，给予附加数据结构，例如：
      *:代表可可执行文件； /:代表目录； =:代表 socket 文件； &#124;:代表 FIFO 文件；
-h  ：将文件大小以人类较易读的方式（例如 GB, KB 等等）列出来；
-i  ：列出 inode 号码，inode 的意义下一章将会介绍；
-l  ：长数据串行出，包含文件的属性与权限等等数据；（常用）
-n  ：列出 UID 与 GID 而非使用者与群组的名称 （UID与GID会在帐号管理提到！）
-r  ：将排序结果反向输出，例如：原本文件名由小到大，反向则为由大到小；
-R  ：连同子目录内容一起列出来，等于该目录下的所有文件都会显示出来；
-S  ：以文件大小大小排序，而不是用文件名排序；
-t  ：依时间排序，而不是用文件名。
--color=never   ：不要依据文件特性给予颜色显示；
--color=always  ：显示颜色
--color=auto    ：让系统自行依据设置来判断是否给予颜色
--full-time     ：以完整时间模式 （包含年、月、日、时、分） 输出
--time={atime,ctime} ：输出 access 时间或改变权限属性时间 （ctime）
                       而非内容变更时间 （modification time）
```

在Linux系统当中，这个 ls 指令可能是最常被执行的吧！因为我们随时都要知道文件或者是目录的相关信息啊～ 不过，我们Linux的文件所记录的信息实在是太多了，ls 没有需要全部都列出来呢～ 所以，当你只有下达 ls 时，默认显示的只有：非隐藏文件的文件名、 以文件名进行排序及文件名代表的颜色显示如此而已。举例来说， 你下达“ ls /etc ”之后，只有经过排序的文件名以及以蓝色显示目录及白色显示一般文件，如此而已。

那如果我还想要加入其他的显示信息时，可以加入上头提到的那些有用的选项呢～ 举例来说，我们之前一直用到的 -l 这个长串显示数据内容，以及将隐藏文件也一起列示出来的 -a 选项等等。 下面则是一些常用的范例，实际试做看看：

```
范例一：将主文件夹下的所有文件列出来（含属性与隐藏文件）
[root@study ~]# ls -al ~
total 56
dr-xr-x---.  5 root root 4096 Jun  4 19:49 .
dr-xr-xr-x. 17 root root 4096 May  4 17:56 ..
-rw-------.  1 root root 1816 May  4 17:57 anaconda-ks.cfg
-rw-------.  1 root root 6798 Jun  4 19:53 .bash_history
-rw-r--r--.  1 root root   18 Dec 29  2013 .bash_logout
-rw-r--r--.  1 root root  176 Dec 29  2013 .bash_profile
-rw-rw-rw-.  1 root root  176 Dec 29  2013 .bashrc
-rw-r--r--.  1 root root  176 Jun  3 00:04 .bashrc_test
drwx------.  4 root root   29 May  6 00:14 .cache
drwxr-xr-x.  3 root root   17 May  6 00:14 .config
# 这个时候你会看到以 . 为开头的几个文件，以及目录档 (.) (..) .config 等等，
# 不过，目录档文件名都是以深蓝色显示，有点不容易看清楚就是了。

范例二：承上题，不显示颜色，但在文件名末显示出该文件名代表的类型（type）
[root@study ~]# ls -alF --color=never  ~
total 56
dr-xr-x---.  5 root root 4096 Jun  4 19:49 ./
dr-xr-xr-x. 17 root root 4096 May  4 17:56 ../
-rw-------.  1 root root 1816 May  4 17:57 anaconda-ks.cfg
-rw-------.  1 root root 6798 Jun  4 19:53 .bash_history
-rw-r--r--.  1 root root   18 Dec 29  2013 .bash_logout
-rw-r--r--.  1 root root  176 Dec 29  2013 .bash_profile
-rw-rw-rw-.  1 root root  176 Dec 29  2013 .bashrc
-rw-r--r--.  1 root root  176 Jun  3 00:04 .bashrc_test
drwx------.  4 root root   29 May  6 00:14 .cache/
drwxr-xr-x.  3 root root   17 May  6 00:14 .config/
# 注意看到显示结果的第一行，嘿嘿～知道为何我们会下达类似 ./command
# 之类的指令了吧？因为 ./ 代表的是“目前目录下”的意思啊！至于什么是 FIFO/Socket ？
# 请参考前一章节的介绍啊！另外，那个.bashrc 时间仅写2013，能否知道详细时间？

范例三：完整的呈现文件的修改时间 (modification time)
[root@study ~]# ls -al --full-time  ~
total 56
dr-xr-x---.  5 root root 4096 2015-06-04 19:49:54.520684829 +0800 .
dr-xr-xr-x. 17 root root 4096 2015-05-04 17:56:38.888000000 +0800 ..
-rw-------.  1 root root 1816 2015-05-04 17:57:02.326000000 +0800 anaconda-ks.cfg
-rw-------.  1 root root 6798 2015-06-04 19:53:41.451684829 +0800 .bash_history
-rw-r--r--.  1 root root   18 2013-12-29 10:26:31.000000000 +0800 .bash_logout
-rw-r--r--.  1 root root  176 2013-12-29 10:26:31.000000000 +0800 .bash_profile
-rw-rw-rw-.  1 root root  176 2013-12-29 10:26:31.000000000 +0800 .bashrc
-rw-r--r--.  1 root root  176 2015-06-03 00:04:16.916684829 +0800 .bashrc_test
drwx------.  4 root root   29 2015-05-06 00:14:56.960764950 +0800 .cache
drwxr-xr-x.  3 root root   17 2015-05-06 00:14:56.975764950 +0800 .config
# 请仔细看，上面的“时间”字段变了喔！变成较为完整的格式。
# 一般来说， ls -al 仅列出目前短格式的时间，有时不会列出年份，
# 借由 --full-time 可以查阅到比较正确的完整时间格式啊！
```

其实 ls 的用法还有很多，包括查阅文件所在 i-node 号码的 ls -i 选项，以及用来进行文件排序的 -S 选项，还有用来查阅不同时间的动作的 --time=atime 等选项（更多时间说明请参考本章后面touch的说明）。而这些选项的存在都是因为 Linux 文件系统记录了很多有用的信息的缘故。那么 Linux 的文件系统中，这些与权限、属性有关的数据放在哪里呢？ 放在 i-node 里面。关于这部分，我们会在下一章继续为你作比较深入的介绍啊！

无论如何， ls 最常被使用到的功能还是那个 -l 的选项，为此，很多 distribution 在默认的情况中， 已经将 ll （L 的小写） 设置成为 ls -l 的意思了！其实，那个功能是 Bash shell 的 alias 功能呢～也就是说，我们直接输入 ll 就等于是输入 ls -l 是一样的～关于这部分，我们会在后续 bash shell 时再次的强调滴～

6.2.2 复制、删除与移动： cp, rm, mv

要复制文件，请使用 cp （copy） 这个指令即可～不过， cp 这个指令的用途可多了～ 除了单纯的复制之外，还可以创建链接文件 （就是捷径啰），比对两文件的新旧而予以更新， 以及复制整个目录等等的功能呢！至于移动目录与文件，则使用 mv （move）， 这个指令也可以直接拿来作更名 （rename） 的动作喔！至于移除吗？那就是 rm （remove） 这个指令啰～下面我们就来瞧一瞧先～

- cp （复制文件或目录）

```
[root@study ~]# cp [-adfilprsu] 来源文件（source） 目标文件（destination）
[root@study ~]# cp [options] source1 source2 source3 .... directory
选项与参数：
-a  ：相当于 -dr --preserve=all 的意思，至于 dr 请参考下列说明；（常用）
-d  ：若来源文件为链接文件的属性（link file），则复制链接文件属性而非文件本身；
-f  ：为强制（force）的意思，若目标文件已经存在且无法打开，则移除后再尝试一次；
-i  ：若目标文件（destination）已经存在时，在覆盖时会先询问动作的进行（常用）
-l  ：进行硬式链接（hard link）的链接文件创建，而非复制文件本身；
-p  ：连同文件的属性（权限、用户、时间）一起复制过去，而非使用默认属性（备份常用）；
-r  ：递回持续复制，用于目录的复制行为；（常用）
-s  ：复制成为符号链接文件 （symbolic link），亦即“捷径”文件；
-u  ：destination 比 source 旧才更新 destination，或 destination 不存在的情况下才复制。
--preserve=all ：除了 -p 的权限相关参数外，还加入 SELinux 的属性, links, xattr 等也复制了。
最后需要注意的，如果来源文件有两个以上，则最后一个目的文件一定要是“目录”才行！
```

复制（cp）这个指令是非常重要的，不同身份者执行这个指令会有不同的结果产生，尤其是那个-a, -p的选项， 对于不同身份来说，差异则非常的大！下面的练习中，有的身份为root有的身份为一般帐号 （在我这里用 dmtsai 这个帐号）， 练习时请特别注意身份的差别喔！
好！开始来做复制的练习与观察：

```
范例一：用root身份，将主文件夹下的 .bashrc 复制到 /tmp 下，并更名为 bashrc
[root@study ~]# cp ~/.bashrc /tmp/bashrc
[root@study ~]# cp -i ~/.bashrc /tmp/bashrc
cp: overwrite `/tmp/bashrc'? n  &lt;==n不覆盖，y为覆盖
# 重复作两次动作，由于 /tmp 下面已经存在 bashrc 了，加上 -i 选项后，
# 则在覆盖前会询问使用者是否确定！可以按下 n 或者 y 来二次确认呢！

范例二：变换目录到/tmp，并将/var/log/wtmp复制到/tmp且观察属性：
[root@study ~]# cd /tmp
[root@study tmp]# cp /var/log/wtmp . &lt;==想要复制到目前的目录，最后的 . 不要忘
[root@study tmp]# ls -l /var/log/wtmp wtmp
-rw-rw-r--. 1 root utmp 28416 Jun 11 18:56 /var/log/wtmp
-rw-r--r--. 1 root root 28416 Jun 11 19:01 wtmp
# 注意上面的特殊字体，在不加任何选项的情况下，文件的某些属性/权限会改变；
# 这是个很重要的特性！要注意喔！还有，连文件创建的时间也不一样了！
# 那如果你想要将文件的所有特性都一起复制过来该怎办？可以加上 -a 喔！如下所示：

[root@study tmp]# cp -a /var/log/wtmp wtmp_2
[root@study tmp]# ls -l /var/log/wtmp wtmp_2
-rw-rw-r--. 1 root utmp 28416 Jun 11 18:56 /var/log/wtmp
-rw-rw-r--. 1 root utmp 28416 Jun 11 18:56 wtmp_2
# 瞭了吧！整个数据特性完全一模一样ㄟ！真是不赖～这就是 -a 的特性！
```

这个 cp 的功能很多，由于我们常常会进行一些数据的复制，所以也会常常用到这个指令的。 一般来说，我们如果去复制别人的数据 （当然，该文件你必须要有 read 的权限才行啊！ ^^） 时， 总是希望复制到的数据最后是我们自己的，所以，在默认的条件中， *cp* 的来源文

件与目的文件的权限是不同的，目的文件的拥有者通常会是指令操作者本身。举例来说， 上面的范例二中，由于我是 *root* 的身份，因此复制过来的文件拥有者与群组就改变成为 *root* 所有了！ 这样说，可以明白吗？^^

由于具有这个特性，因此当我们在进行备份的时候，某些需要特别注意的特殊权限文件， 例如密码档 （/etc/shadow） 以及一些配置文件，就不能直接以 cp 来复制，而必须要加上 -a 或者是 -p 等等可以完整复制文件权限的选项才行！另外，如果你想要复制文件给其他的使用者， 也必须要注意到文件的权限（包含读、写、执行以及文件拥有者等等）， 否则，其他人还是无法针对你给予的文件进行修订的动作喔！注意注意！

```
范例三：复制 /etc/ 这个目录下的所有内容到 /tmp 下面
[root@study tmp]# cp /etc/ /tmp
cp: omitting directory `/etc'   &lt;== 如果是目录则不能直接复制，要加上 -r 的选项
[root@study tmp]# cp -r /etc/ /tmp
# 还是要再次的强调喔！ -r 是可以复制目录，但是，文件与目录的权限可能会被改变
# 所以，也可以利用“ cp -a /etc /tmp ”来下达指令喔！尤其是在备份的情况下！

范例四：将范例一复制的 bashrc 创建一个链接文件 （symbolic link）
[root@study tmp]# ls -l bashrc
-rw-r--r--. 1 root root 176 Jun 11 19:01 bashrc  &lt;==先观察一下文件情况
[root@study tmp]# cp -s bashrc bashrc_slink
[root@study tmp]# cp -l bashrc bashrc_hlink
[root@study tmp]# ls -l bashrc*
-rw-r--r--. 2 root root 176 Jun 11 19:01 bashrc         &lt;==与原始文件不太一样了！
-rw-r--r--. 2 root root 176 Jun 11 19:01 bashrc_hlink
lrwxrwxrwx. 1 root root   6 Jun 11 19:06 bashrc_slink -&gt; bashrc
```

范例四可有趣了！使用 -l 及 -s 都会创建所谓的链接文件（link file），但是这两种链接文件却有不一样的情况。这是怎么一回事啊？ 那个 -l 就是所谓的实体链接（hard link），至于 -s 则是符号链接（symbolic link）， 简单来说，bashrc_slink 是一个“捷径”，这个捷径会链接到bashrc去！所以你会看到文件名右侧会有个指向（->）的符号！

至于bashrc_hlink文件与bashrc的属性与权限完全一模一样，与尚未进行链接前的差异则是第二栏的link数由1变成2了！ 鸟哥这里先不介绍实体链接，因为实体链接涉及 i-node 的相关知识，我们下一章谈到文件系统（filesystem）时再来讨论这个问题。

```
范例五：若 ~/.bashrc 比 /tmp/bashrc 新才复制过来
[root@study tmp]# cp -u ~/.bashrc /tmp/bashrc
# 这个 -u 的特性，是在目标文件与来源文件有差异时，才会复制的。
# 所以，比较常被用于“备份”的工作当中喔！ ^_^

范例六：将范例四造成的 bashrc_slink 复制成为 bashrc_slink_1 与bashrc_slink_2
[root@study tmp]# cp bashrc_slink bashrc_slink_1
[root@study tmp]# cp -d bashrc_slink bashrc_slink_2
[root@study tmp]# ls -l bashrc bashrc_slink*
-rw-r--r--. 2 root root 176 Jun 11 19:01 bashrc
lrwxrwxrwx. 1 root root   6 Jun 11 19:06 bashrc_slink -&gt; bashrc
-rw-r--r--. 1 root root 176 Jun 11 19:09 bashrc_slink_1            &lt;==与原始文件相同
lrwxrwxrwx. 1 root root   6 Jun 11 19:10 bashrc_slink_2 -&gt; bashrc  &lt;==是链接文件！
# 这个例子也是很有趣喔！原本复制的是链接文件，但是却将链接文件的实际文件复制过来了
# 也就是说，如果没有加上任何选项时，cp复制的是原始文件，而非链接文件的属性！
# 若要复制链接文件的属性，就得要使用 -d 的选项了！如 bashrc_slink_2 所示。

范例七：将主文件夹的 .bashrc 及 .bash_history 复制到 /tmp 下面
[root@study tmp]# cp ~/.bashrc ~/.bash_history /tmp
# 可以将多个数据一次复制到同一个目录去！最后面一定是目录！
```

例题：你能否使用 dmtsai 的身份，完整的复制/var/log/wtmp文件到/tmp下面，并更名为dmtsai_wtmp呢？答：实际做看看的结果如下：

```
[dmtsai@study ~]$ cp -a /var/log/wtmp /tmp/dmtsai_wtmp
[dmtsai@study ~]$ ls -l /var/log/wtmp /tmp/dmtsai_wtmp
-rw-rw-r--. 1 dmtsai dmtsai 28416  6月 11 18:56 /tmp/dmtsai_wtmp
-rw-rw-r--. 1 root   utmp   28416  6月 11 18:56 /var/log/wtmp
```

由于 dmtsai 的身份并不能随意修改文件的拥有者与群组，因此虽然能够复制wtmp的相关权限与时间等属性， 但是与拥有者、群组相关的，原本 dmtsai 身份无法进行的动作，即使加上 -a 选项，也是无法达成完整复制权限的！

总之，由于 cp 有种种的文件属性与权限的特性，所以，在复制时，你必须要清楚的了解到：

- 是否需要完整的保留来源文件的信息？
- 来源文件是否为链接文件 （symbolic link file）？
- 来源文件是否为特殊的文件，例如 FIFO, socket 等？
- 来源文件是否为目录？

- rm （移除文件或目录）

```
[root@study ~]# rm [-fir] 文件或目录
选项与参数：
-f  ：就是 force 的意思，忽略不存在的文件，不会出现警告讯息；
-i  ：互动模式，在删除前会询问使用者是否动作
-r  ：递回删除啊！最常用在目录的删除了！这是非常危险的选项！！！

范例一：将刚刚在 cp 的范例中创建的 bashrc 删除掉！
[root@study ~]# cd /tmp
[root@study tmp]# rm -i bashrc
rm: remove regular file `bashrc'? y
# 如果加上 -i 的选项就会主动询问喔，避免你删除到错误的文件名！

范例二：通过万用字符*的帮忙，将/tmp下面开头为bashrc的文件名通通删除：
[root@study tmp]# rm -i bashrc*
# 注意那个星号，代表的是 0 到无穷多个任意字符喔！很好用的东西！

范例三：将 cp 范例中所创建的 /tmp/etc/ 这个目录删除掉！
[root@study tmp]# rmdir /tmp/etc
rmdir: failed to remove '/tmp/etc': Directory not empty   &lt;== 删不掉啊！因为这不是空的目录
[root@study tmp]# rm -r /tmp/etc
rm: descend into directory `/tmp/etc'? y
rm: remove regular file `/tmp/etc/fstab'? y
rm: remove regular empty file `/tmp/etc/crypttab'? ^C  &lt;== 按下 [crtl]+c 中断
.....（中间省略）.....
# 因为身份是 root ，默认已经加入了 -i 的选项，所以你要一直按 y 才会删除！
# 如果不想要继续按 y ，可以按下“ [ctrl]-c ”来结束 rm 的工作。
# 这是一种保护的动作，如果确定要删除掉此目录而不要询问，可以这样做：
[root@study tmp]# \rm -r /tmp/etc
# 在指令前加上反斜线，可以忽略掉 alias 的指定选项喔！至于 alias 我们在bash再谈！
# 拜托！这个范例很可怕！你不要删错了！删除 /etc 系统是会挂掉的！

范例四：删除一个带有 - 开头的文件
[root@study tmp]# touch ./-aaa-  &lt;==[touch](../Text/index.html#touch)这个指令可以创建空文
[root@study tmp]# ls -l
-rw-r--r--. 1 root   root       0 Jun 11 19:22 -aaa-  &lt;==文件大小为0，所以是空文件
[root@study tmp]# rm -aaa-
rm: invalid option -- 'a'                    &lt;== 因为 "-" 是选项嘛！所以系统误判了！
Try 'rm ./-aaa-' to remove the file `-aaa-'. &lt;== 新的 bash 有给建议的
Try 'rm --help' for more information.
[root@study tmp]# rm ./-aaa-
```

这是移除的指令（remove），要注意的是，通常在Linux系统下，为了怕文件被 root 误杀，所以很多 distributions 都已经默认加入 -i 这个选项了！而如果要连目录下的东西都一起杀掉的话， 例如子目录里面还有子目录时，那就要使用 -r 这个选项了！不过，使用“ rm -r ”这个指令之前，请千万注意了，因为该目录或文件“肯定”会被 root 杀掉！因为系统不会再次询问你是否要砍掉呦！所以那是个超级严重的指令下达呦！ 得特别注意！不过，如果你确定该目录不要了，那么使用 rm -r 来循环杀掉是不错的方式！

另外，范例四也是很有趣的例子，我们在之前就谈过，文件名最好不要使用 "-" 号开头， 因为 "-" 后面接的是选项，因此，单纯的使用“ rm -aaa- ”系统的指令就会误判啦！ 那如果使用后面会谈到的正则表达式时，还是会出问题的！所以，只能用避过首位字符是 "-" 的方法啦！ 就是加上本目录“ ./ ”即可！如果 man rm 的话，其实还有一种方法，那就是“ rm -- -aaa- ”也可以啊！

- mv （移动文件与目录，或更名）

```
[root@study ~]# mv [-fiu] source destination
[root@study ~]# mv [options] source1 source2 source3 .... directory
选项与参数：
-f  ：force 强制的意思，如果目标文件已经存在，不会询问而直接覆盖；
-i  ：若目标文件 (destination) 已经存在时，就会询问是否覆盖！
-u  ：若目标文件已经存在，且 source 比较新，才会更新 (update)

范例一：复制一文件，创建一目录，将文件移动到目录中
[root@study ~]# cd /tmp
[root@study tmp]# cp ~/.bashrc bashrc
[root@study tmp]# mkdir mvtest
[root@study tmp]# mv bashrc mvtest
# 将某个文件移动到某个目录去，就是这样做！

范例二：将刚刚的目录名称更名为 mvtest2
[root@study tmp]# mv mvtest mvtest2 &lt;== 这样就更名了！简单～
# 其实在 Linux 下面还有个有趣的指令，名称为 rename ，
# 该指令专职进行多个文件名的同时更名，并非针对单一文件名变更，与mv不同。请man rename。

范例三：再创建两个文件，再全部移动到 /tmp/mvtest2 当中
[root@study tmp]# cp ~/.bashrc bashrc1
[root@study tmp]# cp ~/.bashrc bashrc2
[root@study tmp]# mv bashrc1 bashrc2 mvtest2
# 注意到这边，如果有多个来源文件或目录，则最后一个目标文件一定是“目录！”
# 意思是说，将所有的数据移动到该目录的意思！
```

这是搬移 (move) 的意思！当你要移动文件或目录的时后，呵呵！这个指令就很重要啦！同样的，你也可以使用 -u (update)来测试新旧文件，看看是否需要搬移啰！ 另外一个用途就是“变更文件名！”，我们可以很轻易的使用 mv 来变更一个文件的文件名呢！不过，在 Linux 才有的指令当中，有个 rename ， 可以用来更改大量文件的文件名，你可以利用 man rename 来查阅一下，也是挺有趣的指令喔！

6.2.3 取得路径的文件名称与目录名称

每个文件的完整文件名包含了前面的目录与最终的文件名，而每个文件名的长度都可以到达 255 个字符耶！ 那么你怎么知道那个是文件名？那个是目录名？嘿嘿！就是利用斜线 (/) 来分辨啊！ 其实，取得文件名或者是目录名称，一般的用途应该是在写程序的时候用来判断之用的啦～ 所以，这部分的指令可以用在第三篇内的 shell scripts 里头喔！ 下面我们简单的以几个范例来谈一谈 basename 与 dirname 的用途！

```
[root@study ~]# basename /etc/sysconfig/network
network         &lt;== 很简单！就取得最后的文件名～
[root@study ~]# dirname /etc/sysconfig/network
/etc/sysconfig  &lt;== 取得的变成目录名了！
```

6.3 文件内容查阅

如果我们要查阅一个文件的内容时，该如何是好呢？这里有相当多有趣的指令可以来分享一下：最常使用的显示文件内容的指令可以说是 cat 与 more 及 less 了！此外，如果我们要查看一个很大型的文件（好几百MB时），但是我们只需要后端的几行字而已，那么该如何是好？呵呵！用 tail 呀，此外，tac 这个指令也可以达到这个目的喔！好了，说说各个指令的用途吧！

- cat 由第一行开始显示文件内容
- tac 从最后一行开始显示，可以看出 tac 是 cat 的倒着写！
- nl 显示的时候，顺道输出行号！
- more 一页一页的显示文件内容
- less 与 more 类似，但是比 more 更好的是，他可以往前翻页！
- head 只看头几行
- tail 只看尾巴几行
- od 以二进制的方式读取文件内容！

6.3.1 直接检视文件内容

直接查阅一个文件的内容可以使用 cat/tac/nl 这几个指令啊！

- cat （concatenate）

```
[root@study ~]# cat [-AbEnTv]
选项与参数：
-A  ：相当于 -vET 的整合选项，可列出一些特殊字符而不是空白而已；
-b  ：列出行号，仅针对非空白行做行号显示，空白行不标行号！
-E  ：将结尾的断行字符 $ 显示出来；
-n  ：打印出行号，连同空白行也会有行号，与 -b 的选项不同；
-T  ：将 [tab] 按键以 ^I 显示出来；
-v  ：列出一些看不出来的特殊字符

范例一：检阅 /etc/issue 这个文件的内容
[root@study ~]# cat /etc/issue
\S
Kernel \r on an \m

范例二：承上题，如果还要加印行号呢？
[root@study ~]# cat -n /etc/issue
     1  \S
     2  Kernel \r on an \m
     3
# 所以这个文件有三行！看到了吧！可以印出行号呢！这对于大文件要找某个特定的行时，有点用处！
# 如果不想要编排空白行的行号，可以使用“cat -b /etc/issue”，自己测试看看：

范例三：将 /etc/man_db.conf 的内容完整的显示出来（包含特殊字符）
[root@study ~]# cat -A /etc/man_db.conf
# $
....（中间省略）....
MANPATH_MAP^I/bin^I^I^I/usr/share/man$
MANPATH_MAP^I/usr/bin^I^I/usr/share/man$
MANPATH_MAP^I/sbin^I^I^I/usr/share/man$
MANPATH_MAP^I/usr/sbin^I^I/usr/share/man$
.....（下面省略）.....
# 上面的结果限于篇幅，鸟哥删除掉很多数据了。另外，输出的结果并不会有特殊字体，
# 鸟哥上面的特殊字体是要让您发现差异点在哪里就是了。基本上，在一般的环境中，
# 使用 [tab] 与空白键的效果差不多，都是一堆空白啊！我们无法知道两者的差别。
# 此时使用 cat -A 就能够发现那些空白的地方是啥鬼东西了！[tab]会以 ^I 表示，
# 断行字符则是以 $ 表示，所以你可以发现每一行后面都是 $ 啊！不过断行字符
# 在Windows/Linux则不太相同，Windows的断行字符是 ^M$ 啰。
# 这部分我们会在[第九章 vim 软件](../Text/index.html#tips_dos)的介绍时，再次的说明到喔！
```

嘿嘿！Linux 里面有“猫”指令？喔！不是的， cat 是 Concatenate （连续） 的简写， 主要的功能是将一个文件的内容连续的印出在屏幕上面！例如上面的例子中，我们将 /etc/issue 印出来！如果加上 -n 或 -b 的话，则每一行前面还会加上行号呦！

鸟哥个人是比较少用 cat 啦！毕竟当你的文件内容的行数超过 40 行以上，嘿嘿！根本来不及在屏幕上看到结果！ 所以，配合等一下要介绍的 more 或者是 less 来执行比较好！此外，如果是一般的 DOS 文件时，就需要特别留意一些奇奇怪怪的符号了， 例如断行与 [tab] 等，要显示出来，就得加入 -A 之类的选项了！

- tac （反向列示）

```
[root@study ~]# tac /etc/issue

Kernel \r on an \m
\S
# 嘿嘿！与刚刚上面的范例一比较，是由最后一行先显示喔！
```

tac 这个好玩了！怎么说呢？详细的看一下， cat 与 tac ，有没有发现呀！对啦！ tac 刚好是将 cat 反写过来，所以他的功能就跟 cat 相反啦， cat 是由“第一行到最后一行连续显示在屏幕上”，而 tac 则是“ 由最后一行到第一行反向在屏幕上显示出来 ”，很好玩吧！

- nl （添加行号打印）

```
[root@study ~]# nl [-bnw] 文件
选项与参数：
-b  ：指定行号指定的方式，主要有两种：
      -b a ：表示不论是否为空行，也同样列出行号（类似 cat -n）；
      -b t ：如果有空行，空的那一行不要列出行号（默认值）；
-n  ：列出行号表示的方法，主要有三种：
      -n ln ：行号在屏幕的最左方显示；
      -n rn ：行号在自己字段的最右方显示，且不加 0 ；
      -n rz ：行号在自己字段的最右方显示，且加 0 ；
-w  ：行号字段的占用的字符数。

范例一：用 nl 列出 /etc/issue 的内容
[root@study ~]# nl /etc/issue
     1  \S
     2  Kernel \r on an \m

# 注意看，这个文件其实有三行，第三行为空白（没有任何字符），
# 因为他是空白行，所以 nl 不会加上行号喔！如果确定要加上行号，可以这样做：

[root@study ~]# nl -b a /etc/issue
     1  \S
     2  Kernel \r on an \m
     3
# 呵呵！行号加上来啰～那么如果要让行号前面自动补上 0 呢？可这样

[root@study ~]# nl -b a -n rz /etc/issue
000001  \S
000002  Kernel \r on an \m
000003
# 嘿嘿！自动在自己字段的地方补上 0 了～默认字段是六位数，如果想要改成 3 位数？

[root@study ~]# nl -b a -n rz -w 3 /etc/issue
001     \S
002     Kernel \r on an \m
003
# 变成仅有 3 位数啰～
```

nl 可以将输出的文件内容自动的加上行号！其默认的结果与 cat -n 有点不太一样， nl 可以将行号做比较多的显示设计，包括位数与是否自动补齐 0 等等的功能呢。

6.3.2 可翻页检视

前面提到的 nl 与 cat, tac 等等，都是一次性的将数据一口气显示到屏幕上面，那有没有可以进行一页一页翻动的指令啊？ 让我们可以一页一页的观察，才不会前面的数据看不到啊～呵呵！有的！那就是 more 与 less 啰～

- more （一页一页翻动）

```
[root@study ~]# more /etc/man_db.conf
#
#
# This file is used by the man-db package to configure the man and cat paths.
# It is also used to provide a manpath for those without one by examining
# their PATH environment variable. For details see the manpath（5） man page.
#
.....（中间省略）.....
--More--（28%）  &lt;== 重点在这一行喔！你的光标也会在这里等待你的指令
```

仔细的给他看到上面的范例，如果 more 后面接的文件内容行数大于屏幕输出的行数时， 就会出现类似上面的图示。重点在最后一行，最后一行会显示出目前显示的百分比， 而且还可以在最后一行输入一些有用的指令喔！在 more 这个程序的运行过程中，你有几个按键可以按的：

- 空白键 （space）：代表向下翻一页；
- Enter ：代表向下翻"一行"；
- /字串 ：代表在这个显示的内容当中，向下搜寻"字串"这个关键字；
- :f ：立刻显示出文件名以及目前显示的行数；
- q ：代表立刻离开 more ，不再显示该文件内容。
- b 或 [ctrl]-b ：代表往回翻页，不过这动作只对文件有用，对管线无用。

要离开 more 这个指令的显示工作，可以按下 q 就能够离开了。而要向下翻页，就使用空白键即可。 比较有用的是搜寻字串的功能，举例来说，我们使用" more /etc/man_db.conf "来观察该文件， 若想要在该文件内搜寻 MANPATH 这个字串时，可以这样做：

```
[root@study ~]# more /etc/man_db.conf
#
#
# This file is used by the man-db package to configure the man and cat paths.
# It is also used to provide a manpath for those without one by examining
# their PATH environment variable. For details see the manpath（5） man page.
#
....（中间省略）....
/MANPATH   &lt;== 输入了 / 之后，光标就会自动跑到最下面一行等待输入！
```

如同上面的说明，输入了 / 之后，光标就会跑到最下面一行，并且等待你的输入， 你输入了字串并按下[enter]之后，嘿嘿！ more 就会开始向下搜寻该字串啰～而重复搜寻同一个字串，可以直接按下 n 即可啊！最后，不想要看了，就按下 q 即可离开 more 啦！

- less （一页一页翻动）

```
[root@study ~]# less /etc/man_db.conf
#
#
# This file is used by the man-db package to configure the man and cat paths.
# It is also used to provide a manpath for those without one by examining
# their PATH environment variable. For details see the manpath（5） man page.
#
.....（中间省略）.....
:   &lt;== 这里可以等待你输入指令！
```

less 的用法比起 more 又更加的有弹性，怎么说呢？在 more 的时候，我们并没有办法向前面翻， 只能往后面看，但若使用了 less 时，呵呵！就可以使用 [pageup] [pagedown] 等按键的功能来往前往后翻看文件，你瞧，是不是更容易使用来观看一个文件的内容了呢！

除此之外，在 less 里头可以拥有更多的"搜寻"功能喔！不止可以向下搜寻，也可以向上搜寻～ 实在是很不错用～基本上，可以输入的指令有：

- 空白键 ：向下翻动一页；

- [pagedown]：向下翻动一页；
- [pageup] ：向上翻动一页；
- /字串 ：向下搜寻“字串”的功能；
- ?字串 ：向上搜寻“字串”的功能；
- n ：重复前一个搜寻 （与 / 或 ? 有关！）
- N ：反向的重复前一个搜寻 （与 / 或 ? 有关！）
- g ：前进到这个数据的第一行去；
- G ：前进到这个数据的最后一行去 （注意大小写）；
- q ：离开 less 这个程序；

查阅文件内容还可以进行搜寻的动作～瞧～ less 是否很不错用啊！ 其实 less 还有很多的功能喔！详细的使用方式请使用 man less 查询一下啊！ ^_^

你是否会觉得 less 使用的画面与环境与 man page 非常的类似呢？没错啦！因为man这个指令就是调用 less 来显示说明文档的内容的！ 现在你是否觉得 less 很重要呢？ ^_^

6.3.3 数据撷取

我们可以将输出的数据作一个最简单的撷取，那就是取出文件前面几行 （head） 或取出后面几行 （tail） 文字的功能。 不过，要注意的是， head 与 tail 都是以“行”为单位来进行数据撷取的喔！

- head （取出前面几行）

```
[root@study ~]# head [-n number] 文件
选项与参数：
-n  ：后面接数字，代表显示几行的意思

[root@study ~]# head /etc/man_db.conf
# 默认的情况中，显示前面十行！若要显示前 20 行，就得要这样：
[root@study ~]# head -n 20 /etc/man_db.conf

范例：如果后面100行的数据都不打印，只打印/etc/man_db.conf的前面几行，该如何是好？
[root@study ~]# head -n -100 /etc/man_db.conf
```

head 的英文意思就是“头”啦，那么这个东西的用法自然就是显示出一个文件的前几行啰！ 没错！就是这样！若没有加上 -n 这个选项时，默认只显示十行，若只要一行呢？那就加入“ head -n 1 filename ”即可！

另外那个 -n 选项后面的参数较有趣，如果接的是负数，例如上面范例的-n -100时，代表列前的所有行数， 但不包括后面100行。举例来说 CentOS 7.1 的 /etc/man*db.conf 共有131行，则上述的指令“head -n -100 /etc/man_db.conf” 就会列出前面31行，后面100行不会打印出来了。这样说，比较容易懂了吧？ ^^*

- tail （取出后面几行）

```
[root@study ~]# tail [-n number] 文件
选项与参数：
-n  ：后面接数字，代表显示几行的意思
-f  ：表示持续侦测后面所接的文件名，要等到按下[ctrl]-c才会结束tail的侦测

[root@study ~]# tail /etc/man_db.conf
# 默认的情况中，显示最后的十行！若要显示最后的 20 行，就得要这样：
[root@study ~]# tail -n 20 /etc/man_db.conf

范例一：如果不知道/etc/man_db.conf有几行，却只想列出100行以后的数据时？
[root@study ~]# tail -n +100 /etc/man_db.conf

范例二：持续侦测/var/log/messages的内容
[root@study ~]# tail -f /var/log/messages
  &lt;==要等到输入[crtl]-c之后才会离开tail这个指令的侦测！
```

有 head 自然就有 tail （ 尾巴 ） 啰！没错！这个 tail 的用法跟 head 的用法差不多类似，只是显示的是后面几行就是了！默认也是显示十行，若要显示非十行，就加 -n number 的选项即可。

范例一的内容就有趣啦！其实与head -n -xx有异曲同工之妙。当下达“tail -n +100 /etc/man_db.conf” 代表该文件从100行以后都会被列出来，同样的，在man_db.conf共有131行，因此第100~131行就会被列出来啦！ 前面的99行都不会被显示出来喔！

至于范例二中，由于/var/log/messages随时会有数据写入，你想要让该文件有数据写入时就立刻显示到屏幕上， 就利用 -f 这个选项，他可以一直侦测/var/log/messages这个文件，新加入的数据都会被显示到屏幕上。 直到你按下[crtl]-c才会离开tail的侦测喔！由于 messages 必须要 root 权限才能看，所以该范例得要使用 root 来查询喔！

例题：假如我想要显示 /etc/man_db.conf 的第 11 到第 20 行呢？答：这个应该不算难，想一想，在第 11 到第 20 行，那么我取前 20 行，再取后十行，所以结果就是：“ head -n 20 /etc/man_db.conf | tail -n 10 ”，这样就可以得到第 11 到第 20 行之间的内容了！

这两个指令中间有个管线 （|） 的符号存在，这个管线的意思是：“前面的指令所输出的讯息，请通过管线交由后续的指令继续使用”的意思。 所以， head -n 20 /etc/man_db.conf 会将文件内的 20 行取出来，但不输出到屏幕上，而是转交给后续的 tail 指令继续处理。 因此 tail “不需要接文件名”，因为 tail 所需要的数据是来自于 head 处理后的结果！这样说，有没有理解？

更多的管线命令，我们会在第三篇继续解释的！

例题：承上一题，那如果我想要列出正确的行号呢？就是屏幕上仅列出 /etc/man_db.conf 的第 11 到第 20 行，且有行号存在？答：我们可以通过 cat -n 来带出行号，然后再通过 head/tail 来撷取数据即可！所以就变成如下的模样了： cat -n /etc/man_db.conf | head -n 20 | tail -n 10

6.3.4 非纯文本文件： od

我们上面提到的，都是在查阅纯文本文件的内容。那么万一我们想要查阅非文本文件，举例来说，例如 /usr/bin/passwd 这个可执行文件的内容时， 又该如何去读出信息呢？事实上，由于可执行文件通常是 binary file ，使用上头提到的指令来读取他的内容时， 确实会产生类似乱码的数据啊！那怎么办？没关系，我们可以利用 od 这个指令来读取喔！

```
[root@study ~]# od [-t TYPE] 文件
选项或参数：
-t  ：后面可以接各种“类型 （TYPE）”的输出，例如：
      a       ：利用默认的字符来输出；
      c       ：使用 ASCII 字符来输出
      d[size] ：利用十进制（decimal）来输出数据，每个整数占用 size Bytes ；
      f[size] ：利用浮点数值（floating）来输出数据，每个数占用 size Bytes ；
      o[size] ：利用八进位（octal）来输出数据，每个整数占用 size Bytes ；
      x[size] ：利用十六进制（hexadecimal）来输出数据，每个整数占用 size Bytes ；

范例一：请将/usr/bin/passwd的内容使用ASCII方式来展现！
[root@study ~]# od -t c /usr/bin/passwd
0000000 177   E   L   F 002 001 001  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000020 003  \0   &gt;  \0 001  \0  \0  \0 364   3  \0  \0  \0  \0  \0  \0
0000040   @  \0  \0  \0  \0  \0  \0  \0   x   e  \0  \0  \0  \0  \0  \0
0000060  \0  \0  \0  \0   @  \0   8  \0  \t  \0   @  \0 035  \0 034  \0
0000100 006  \0  \0  \0 005  \0  \0  \0   @  \0  \0  \0  \0  \0  \0  \0
.....（后面省略）....
# 最左边第一栏是以 8 进位来表示Bytes数。以上面范例来说，第二栏0000020代表开头是
# 第 16 个 byes （2x8） 的内容之意。

范例二：请将/etc/issue这个文件的内容以8进位列出储存值与ASCII的对照表
[root@study ~]# od -t oCc /etc/issue
0000000 134 123 012 113 145 162 156 145 154 040 134 162 040 157 156 040
          \   S  \n   K   e   r   n   e   l       \   r       o   n
0000020 141 156 040 134 155 012 012
          a   n       \   m  \n  \n
0000027
# 如上所示，可以发现每个字符可以对应到的数值为何！要注意的是，该数值是 8 进位喔！
# 例如 S 对应的记录数值为 123 ，转成十进制：1x8^2+2x8+3=83。
```

利用这个指令，可以将 data file 或者是 binary file 的内容数据给他读出来喔！ 虽然读出的来数值默认是使用非文本文件，亦即是 16 进位的数值来显示的， 不过，我们还是可以通过 -t c 的选项与参数来将数据内的字符以 ASCII 类型的字符来显示， 虽然对于一般使用者来说，这个指令的用处可能不大，但是对于工程师来说， 这个指令可以将 binary file 的内容作一个大致的输出，他们可以看得出东西的啦～ ^_^

如果对纯文本文件使用这个指令，你甚至可以发现到 ASCII 与字符的对照表！非常有趣！ 例如上述的范例二，你可以发现到每个英文字 S 对照到的数字都是 123，转成十进制你就能够发现那是 83 啰！ 如果你有任何程序语言的书，拿出来对照一下 ASCII 的对照表，就能够发现真是正确啊！呵呵！

例题：我不想找 google，想要立刻找到 password 这几个字的 ASCII 对照，该如何通过 od 来判断？答：其实可以通过刚刚上一个小节谈到的管线命令来处理！如下所示： echo password | od -t oCc echo 可以在屏幕上面显示任何信息，而这个信息不由屏幕输出，而是传给 od 去继续处理！就可以得到 ASCII code 对照啰！

6.3.5 修改文件时间或创建新文件： touch

我们在 ls 这个指令的介绍时，有稍微提到每个文件在linux下面都会记录许多的时间参数， 其实是有三个主要的变动时间，那么三个时间的意义是什么呢？

- **modification time （mtime）**： 当该文件的“内容数据”变更时，就会更新这个时间！内容数据指的是文件的内容，而不是文件的属性或权限喔！
- **status time （ctime）**： 当该文件的“状态 （status）”改变时，就会更新这个时间，举例来说，像是权限与属性被更改了，都会更新这个时间啊。
- **access time （atime）**： 当“该文件的内容被取用”时，就会更新这个读取时间 （access）。举例来说，我们使用 cat 去读取 /etc/man_db.conf ， 就会更新该文件的 atime 了。

这是个挺有趣的现象，举例来说，我们来看一看你自己的 /etc/man_db.conf 这个文件的时间吧！

```
[root@study ~]# date; ls -l /etc/man_db.conf ; ls -l --time=atime /etc/man_db.conf ; \
&gt; ls -l --time=ctime /etc/man_db.conf # 这两行其实是同一行喔！用分号隔开
Tue Jun 16 00:43:17 CST 2015  # 目前的时间啊！
-rw-r--r--. 1 root root 5171 Jun 10  2014 /etc/man_db.conf  # 在 2014/06/10 创建的内容（mti
-rw-r--r--. 1 root root 5171 Jun 15 23:46 /etc/man_db.conf  # 在 2015/06/15 读取过内容（ati
-rw-r--r--. 1 root root 5171 May  4 17:54 /etc/man_db.conf  # 在 2015/05/04 更新过状态（cti
# 为了要让数据输出比较好看，所以鸟哥将三个指令同时依序执行，三个指令中间用分号 （;） 隔开即可。
```

看到了吗？在默认的情况下，ls 显示出来的是该文件的 mtime ，也就是这个文件的内容上次被更动的时间。 至于鸟哥的系统是在 5 月 4 号的时候安装的，因此，这个文件被产生导致状态被更动的时间就回溯到那个时间点了（ctime）！ 而还记得刚刚我们使用的范例当中，有使用到man_db.conf这个文件啊，所以啊，他的 atime 就会变成刚刚使用的时间了！

文件的时间是很重要的，因为，如果文件的时间误判的话，可能会造成某些程序无法顺利的运行。 OK！那么万一我发现了一个文件来自未来，该如何让该文件的时间变成“现在”的时刻呢？ 很简单啊！就用“touch”这个指令即可！



Tips 嘿嘿！不要怀疑系统时间会“来自未来”喔！很多时候会有这个问题的！举例来说在安装过后系统时间可能会被改变！ 因为台湾时区在国际标准时间“格林威治时间, GMT”的右边，所以会比较早看到阳光，也就是说，台湾时间比GMT时间快了八小时！ 如果安装行为不当，我们的系统可能会有八小时快转，你的文件就有可能来自八小时后了。

至于某些情况下，由于BIOS的设置错误，导致系统时间跑到未来时间，并且你又创建了某些文件。 等你将时间改回正确的时间时，该文件不就变成来自未来了？^_^

```
[root@study ~]# touch [-acdmt] 文件
选项与参数：
-a  ：仅修订 access time；
-c  ：仅修改文件的时间，若该文件不存在则不创建新文件；
-d  ：后面可以接欲修订的日期而不用目前的日期，也可以使用 --date="日期或时间"
-m  ：仅修改 mtime ；
-t  ：后面可以接欲修订的时间而不用目前的时间，格式为[YYYYMMDDhhmm]

范例一：新建一个空的文件并观察时间
[dmtsai@study ~]# cd /tmp
[dmtsai@study tmp]# touch testtouch
[dmtsai@study tmp]# ls -l testtouch
-rw-rw-r--. 1 dmtsai dmtsai 0 Jun 16 00:45 testtouch
# 注意到，这个文件的大小是 0 呢！在默认的状态下，如果 touch 后面有接文件，
# 则该文件的三个时间 （atime/ctime/mtime） 都会更新为目前的时间。若该文件不存在，
# 则会主动的创建一个新的空的文件喔！例如上面这个例子！

范例二：将 ~/.bashrc 复制成为 bashrc，假设复制完全的属性，检查其日期
[dmtsai@study tmp]# cp -a ~/.bashrc bashrc
[dmtsai@study tmp]# date; ll bashrc; ll --time=atime bashrc; ll --time=ctime bashrc
Tue Jun 16 00:49:24 CST 2015                          &lt;==这是目前的时间
-rw-r--r--. 1 dmtsai dmtsai 231 Mar  6 06:06 bashrc  &lt;==这是 mtime
-rw-r--r--. 1 dmtsai dmtsai 231 Jun 15 23:44 bashrc  &lt;==这是 atime
-rw-r--r--. 1 dmtsai dmtsai 231 Jun 16 00:47 bashrc  &lt;==这是 ctime
```

在上面这个案例当中我们使用了“ll”这个指令（两个英文L的小写），这个指令其实就是“ls -l”的意思， ll本身不存在，是被“做出来”的一个命令别名。相关的命令别名我们会在bash章节当中详谈的，这里先知道ll="ls -l"即可。 至于分号“；”则代表连续指令的下达啦！你可以在一行指令当中写入多重指令， 这些指令可以“依序”执行。由上面的指令我们会知道ll那一行有三个指令被下达在同一行中。

至于执行的结果当中，我们可以发现数据的内容与属性是被复制过来的，因此文件内容时间（mtime）与原本文件相同。 但是由于这个文件是刚刚被创建的，因此状态（ctime）就变成现在的时间啦！那如果你想要变更这个文件的时间呢？可以这样做：

```
范例三：修改案例二的 bashrc 文件，将日期调整为两天前
[dmtsai@study tmp]# touch -d "2 days ago" bashrc
[dmtsai@study tmp]# date; ll bashrc; ll --time=atime bashrc; ll --time=ctime bashrc
Tue Jun 16 00:51:52 CST 2015
-rw-r--r--. 1 dmtsai dmtsai 231 Jun 14 00:51 bashrc
-rw-r--r--. 1 dmtsai dmtsai 231 Jun 14 00:51 bashrc
-rw-r--r--. 1 dmtsai dmtsai 231 Jun 16 00:51 bashrc
# 跟上个范例比较看看，本来是 16 日变成 14 日了 （atime/mtime）～不过， ctime 并没有跟着改变喔！

范例四：将上个范例的 bashrc 日期改为 2014/06/15 2:02
[dmtsai@study tmp]# touch -t 201406150202 bashrc
[dmtsai@study tmp]# date; ll bashrc; ll --time=atime bashrc; ll --time=ctime bashrc
Tue Jun 16 00:54:07 CST 2015
-rw-r--r--. 1 dmtsai dmtsai 231 Jun 15  2014 bashrc
-rw-r--r--. 1 dmtsai dmtsai 231 Jun 15  2014 bashrc
-rw-r--r--. 1 dmtsai dmtsai 231 Jun 16 00:54 bashrc
# 注意看看，日期在 atime 与 mtime 都改变了，但是 ctime 则是记录目前的时间！
```

通过 touch 这个指令，我们可以轻易的修订文件的日期与时间。并且也可以创建一个空的文件喔！ 不过，要注意的是，即使我们复制一个文件时，复制所有的属性，但也没有办法复制 ctime 这个属性的。 ctime 可以记录这个文件最近的状态 （status） 被改变的时间。无论如

何，还是要告知大家， 我们平时看的文件属性中，比较重要的还是属于那个 mtime 啊！我们关心的常常是这个文件的“内容” 是什么时候被更动的说～瞭乎？

无论如何， touch 这个指令最常被使用的情况是：

- 创建一个空的文件；
- 将某个文件日期修订为目前 （mtime 与 atime）

6.4 文件与目录的默认权限与隐藏权限

由第五章、Linux文件权限的内容我们可以知道一个文件有若干个属性， 包括读写执行（r, w, x）等基本权限，及是否为目录 （d） 与文件 （-） 或者是链接文件 （l） 等等的属性！ 要修改属性的方法在前面也约略提过了（chgrp, chown, chmod） ，本小节会再加强补充一下！

除了基本r, w, x权限外，在Linux传统的Ext2/Ext3/Ext4文件系统下，我们还可以设置其他的系统隐藏属性， 这部份可使用 chattr 来设置，而以 lsattr 来查看，最重要的属性就是可以设置其不可修改的特性！让连文件的拥有者都不能进行修改！ 这个属性可是相当重要的，尤其是在安全机制上面 （security）！比较可惜的是，在 CentOS 7.x 当中利用 xfs 作为默认文件系统， 但是 xfs 就没有支持所有的 chattr 的参数了！仅有部份参数还有支持而已！

首先，先来复习一下上一章谈到的权限概念，将下面的例题看一看先：

例题：你的系统有个一般身份使用者 dmtsai，他的群组属于 dmtsai，他的主文件夹在 /home/dmtsai， 你是root，你想将你的 ~/.bashrc 复制给他，可以怎么作？答：由上一章的权限概念我们可以知道 root 虽然可以将这个文件复制给 dmtsai，不过这个文件在 dmtsai 的主文件夹中却可能让 dmtsai 没有办法读写（因为该文件属于 root 的嘛！而 dmtsai 又不能使用 chown 之故）。 此外，我们又担心覆盖掉 dmtsai 自己的 .bashrc 配置文件，因此，我们可以进行如下的动作喔： 复制文件： cp ~/.bashrc ~dmtsai/bashrc 修改属性： chown dmtsai:dmtsai ~dmtsai/bashrc

例题：我想在 /tmp 下面创建一个目录，这个目录名称为 chapter6_1 ，并且这个目录拥有者为 dmtsai， 群组为 dmtsai，此外，任何人都可以进入该目录浏览文件，不过除了 dmtsai 之外，其他人都不能修改该目录下的文件。答：因为除了 dmtsai 之外，其他人不能修改该目录下的文件，所以整个目录的权限应该是 drwxr-xr-x 才对！ 因此你应该这样做：创建目录： mkdir /tmp/chapter6_1 修改属性： chown -R dmtsai:dmtsai /tmp/chapter6_1 修改权限： chmod -R 755 /tmp/chapter6_1

在上面这个例题当中，如果你知道 755 那个分数是怎么计算出来的，那么你应该对于权限有一定程度的概念了。 如果你不知道 755 怎么来的？那么...赶快回去前一章看看 chmod 那个指令的介绍部分啊！这部分很重要喔！你得要先清楚的了解到才行～否则就进行不下去啰～ 假设你对于权限都认识的差不多了，那么下面我们就要来谈一谈，“新增一个文件或目录时，默认的权限是什么？”这个议题！

6.4.1 文件默认权限：umask

OK！那么现在我们知道如何创建或者是改变一个目录或文件的属性了，不过， 你知道当你创建一个新的文件或目录时，他的默认权限会是什么吗？呵呵！那就与 umask 这个玩意儿有关了！那么 umask 是在搞什么呢？基本上， umask 就是指定 “目前使用者在创建文件或目录时候的权限默认值”， 那么如何得知或设置 umask 呢？他的指定条件以下面的方式来指定：

```
[root@study ~]# umask
0022              &lt;==与一般权限有关的是后面三个数字！
[root@study ~]# umask -S
u=rwx,g=rx,o=rx
```

查阅的方式有两种，一种可以直接输入 umask ，就可以看到数字体态的权限设置分数， 一种则是加入 -S （Symbolic） 这个选项，就会以符号类型的方式来显示出权限了！ 奇怪的是，怎么 umask 会有四组数字啊？不是只有三组吗？是没错啦。 第一组是特殊权限用的，我们先不要理他，所以先看后面三组即可。

在默认权限的属性上，目录与文件是不一样的。从第五章我们知道 x 权限对于目录是非常重要的！ 但是一般文件的创建则不应该有执行的权限，因为一般文件通常是用在于数据的记录嘛！当然不需要执行的权限了。 因此，默认的情况如下：

- 若使用者创建为“文件”则默认“没有可执行（ x ）权限”，亦即只有 rw 这两个项目，也就是最大为 666 分，默认权限如下： -rw-rw-rw-
- 若使用者创建为“目录”，则由于 x 与是否可以进入此目录有关，因此默认为所有权限均开放，亦即为 777 分，默认权限如下： drwxrwxrwx

要注意的是，umask 的分数指的是“该默认值需要减掉的权限！”因为 r、w、x 分别是 4、2、1 分，所以啰！也就是说，当要拿掉能写的权限，就是输入 2 分，而如果要拿掉能读的权限，也就是 4 分，那么要拿掉读与写的权限，也就是 6 分，而要拿掉执行与写入的权限，也就是 3 分，这样了解吗？请问你， 5 分是什么？呵呵！ 就是读与执行的权限啦！

如果以上面的例子来说明的话，因为 umask 为 022 ，所以 user 并没有被拿掉任何权限，不过 group 与 others 的权限被拿掉了 2 （也就是 w 这个权限），那么当使用者：

- 创建文件时：（-rw-rw-rw-） - （-----w--w-） ==> -rw-r--r--
- 创建目录时：（drwxrwxrwx） - （d----w--w-） ==> drwxr-xr-x

不相信吗？我们就来测试看看吧！

```
[root@study ~]# umask
0022
[root@study ~]# touch test1
[root@study ~]# mkdir test2
[root@study ~]# ll -d test*
-rw-r--r--. 1 root root 0  6月 16 01:11 test1
drwxr-xr-x. 2 root root 6  6月 16 01:11 test2
```

呵呵！瞧见了吧！确定新建文件的权限是没有错的。

- umask的利用与重要性：专题制作

想像一个状况，如果你跟你的同学在同一部主机里面工作时，因为你们两个正在进行同一个专题， 老师也帮你们两个的帐号创建好了相同群组的状态，并且将 /home/class/ 目录做为你们两个人的专题目录。 想像一下，有没有可能你所制作的文件你的同学无法编辑？果真如此

的话，那就伤脑筋了！

这个问题很常发生啊！举上面的案例来看就好了，你看一下 test1 的权限是几分？ 644 呢！意思是"如果 umask 订定为 022 ，那新建的数据只有使用者自己具有 w 的权限， 同群组的人只有 r 这个可读的权限而已，并无法修改喔！"这样要怎么共同制作专题啊！您说是吧！

所以，当我们需要新建文件给同群组的使用者共同编辑时，那么 umask 的群组就不能拿掉 2 这个 w 的权限！ 所以啰， umask 就得要是 002 之类的才可以！这样新建的文件才能够是 -rw-rw-r-- 的权限模样喔！ 那么如何设置 umask 呢？简单的很，直接在 umask 后面输入 002 就好了！

```
[root@study ~]# umask 002
[root@study ~]# touch test3
[root@study ~]# mkdir test4
[root@study ~]# ll -d test[34]   # 中括号 [ ] 代表中间有个指定的字符，而不是任意字符的意思
-rw-rw-r--. 1 root root 0  6月 16 01:12 test3
drwxrwxr-x. 2 root root 6  6月 16 01:12 test4
```

所以说，这个 umask 对于新建文件与目录的默认权限是很有关系的！这个概念可以用在任何服务器上面， 尤其是未来在你架设文件服务器 （file server） ，举例来说， SAMBA Server 或者是 FTP server 时， 都是很重要的观念！这牵涉到你的使用者是否能够将文件进一步利用的问题喔！不要等闲视之！

例题：假设你的 umask 为 003 ，请问该 umask 情况下，创建的文件与目录权限为？答：umask 为 003 ，所以拿掉的权限为 --------wx，因此： 文件： （-rw-rw-rw-） - （--------wx） = -rw-rw-r-- 目录： （drwxrwxrwx） - （d-------wx） = drwxrwxr--



Tips 关于 umask 与权限的计算方式中，教科书喜欢使用二进制的方式来进行 AND 与 NOT 的计算， 不过，鸟哥还是比较喜欢使用符号方式来计算～联想上面比较容易一点～

但是，有的书籍或者是 BBS 上面的朋友，喜欢使用文件默认属性 666 与目录默认属性 777 来与 umask 进行相减的计算～这是不好的喔！以上面例题来看， 如果使用默认属性相加减，则文件变成：666-003=663，亦即是 -rw-rw--wx ，这可是完全不对的喔！ 想想看，原本文件就已经去除 x 的默认属性了，怎么可能突然间冒出来了？ 所以，这个地方得要特别小心喔！

在默认的情况中， root 的 umask 会拿掉比较多的属性，root 的 umask 默认是 022 ， 这是基于安全的考虑啦～至于一般身份使用者，通常他们的 umask 为 002 ，亦即保留同群组的写入权力！ 其实，关于默认 umask 的设置可以参考 /etc/bashrc 这个文件的内容，不过，不建议修改该文件， 你可以参考第十章 bash shell 提到的环境参数配置文件 （~/.bashrc） 的说明！

6.4.2 文件隐藏属性

什么？文件还有隐藏属性？光是那九个权限就快要疯掉了，竟然还有隐藏属性，眞是要命～但是没办法，就是有文件的隐藏属性存在啊！不过，这些隐藏的属性确实对于系统有很大的帮助的～ 尤其是在系统安全 （Security） 上面，重要的紧呢！不过要先强调的是，下面的chattr指令只能在Ext2/Ext3/Ext4的 Linux 传统文件系统上面完整生效， 其他的文件系统可能就无法完整的支持这个指令了，例如 xfs 仅支持部份参数而已。下面我们就来谈一谈如何设置与检查这些隐藏的属性吧！

- chattr （设置文件隐藏属性）

```
[root@study ~]# chattr [+-=][ASacdistu] 文件或目录名称
选项与参数：
+   ：增加某一个特殊参数，其他原本存在参数则不动。
-   ：移除某一个特殊参数，其他原本存在参数则不动。
=   ：设置一定，且仅有后面接的参数

A  ：当设置了 A 这个属性时，若你有存取此文件（或目录）时，他的存取时间 atime 将不会被修改，
     可避免 I/O 较慢的机器过度的存取磁盘。（目前建议使用文件系统挂载参数处理这个项目）
S  ：一般文件是非同步写入磁盘的（原理请参考[前一章sync](../Text/index.html#sync)的说明），如果加上 S
     当你进行任何文件的修改，该更动会“同步”写入磁盘中。
a  ：当设置 a 之后，这个文件将只能增加数据，而不能删除也不能修改数据，只有root 才能设置这属性
c  ：这个属性设置之后，将会自动的将此文件“压缩”，在读取的时候将会自动解压缩，
     但是在储存的时候，将会先进行压缩后再储存（看来对于大文件似乎蛮有用的！）
d  ：当 dump 程序被执行的时候，设置 d 属性将可使该文件（或目录）不会被 dump 备份
i  ：这个 i 可就很厉害了！他可以让一个文件“不能被删除、改名、设置链接也无法写入或新增数据！”
     对于系统安全性有相当大的助益！只有 root 能设置此属性
s  ：当文件设置了 s 属性时，如果这个文件被删除，他将会被完全的移除出这个硬盘空间，
     所以如果误删了，完全无法救回来了喔！
u  ：与 s 相反的，当使用 u 来设置文件时，如果该文件被删除了，则数据内容其实还存在磁盘中，
     可以使用来救援该文件喔！
注意1：属性设置常见的是 a 与 i 的设置值，而且很多设置值必须要身为 root 才能设置
注意2：xfs 文件系统仅支持 AadiS 而已

范例：请尝试到/tmp下面创建文件，并加入 i 的参数，尝试删除看看。
[root@study ~]# cd /tmp
[root@study tmp]# touch attrtest     &lt;==创建一个空文件
[root@study tmp]# chattr +i attrtest &lt;==给予 i 的属性
[root@study tmp]# rm attrtest        &lt;==尝试删除看看
rm: remove regular empty file `attrtest'? y
rm: cannot remove `attrtest': Operation not permitted
# 看到了吗？呼呼！连 root 也没有办法将这个文件删除呢！赶紧解除设置！

范例：请将该文件的 i 属性取消！
[root@study tmp]# chattr -i attrtest
```

这个指令是很重要的，尤其是在系统的数据安全上面！由于这些属性是隐藏的性质，所以需要以 lsattr 才能看到该属性呦！其中，个人认为最重要的当属 +i 与 +a 这个属性了。+i 可以让一个文件无法被更动，对于需要强烈的系统安全的人来说， 眞是相当的重要的！里头还有相当多的属性是需要 root 才能设置的呢！

此外，如果是 log file 这种的登录文件，就更需要 +a 这个可以增加，但是不能修改旧有的数据与删除的参数了！怎样？很棒吧！ 未来提到登录文件 （十八章） 的认知时，我们再来聊一聊如何设置他吧！

- lsattr （显示文件隐藏属性）

```
[root@study ~]# lsattr [-adR] 文件或目录
选项与参数：
-a ：将隐藏文件的属性也秀出来；
-d ：如果接的是目录，仅列出目录本身的属性而非目录内的文件名；
-R ：连同子目录的数据也一并列出来！

[root@study tmp]# chattr +aiS attrtest
[root@study tmp]# lsattr attrtest
--S-ia---------- attrtest
```

使用 chattr 设置后，可以利用 lsattr 来查阅隐藏的属性。不过， 这两个指令在使用上必须要特别小心，否则会造成很大的困扰。例如：某天你心情好，突然将 /etc/shadow 这个重要的密码记录文件给他设置成为具有 i 的属性，那么过了若干天之后， 你突然要新增使用者，却一直无法新增！别怀疑，赶快去将 i 的属性拿掉吧！

6.4.3 文件特殊权限： SUID, SGID, SBIT

我们前面一直提到关于文件的重要权限，那就是 rwx 这三个读、写、执行的权限。 但是，眼尖的朋友们在第五章的目录树章节中， 一定注意到了一件事，那就是，怎么我们的 /tmp 权限怪怪的？ 还有，那个 /usr/bin/passwd 也怪怪的？怎么回事啊？看看先：

```
[root@study ~]# ls -ld /tmp ; ls -l /usr/bin/passwd
drwxrwxrwt. 14 root root 4096 Jun 16 01:27 /tmp
-rwsr-xr-x. 1 root root 27832 Jun 10  2014 /usr/bin/passwd
```

不是应该只有 rwx 吗？还有其他的特殊权限（ s 跟 t ）啊？啊.....头又开始昏了～ @_@ 因为 s 与 t 这两个权限的意义与系统的帐号 （第十三章）及系统的程序（process, 第十六章）较为相关， 所以等到后面的章节谈完后你才会比较有概念！下面的说明先看看就好，如果看不懂也没有关系， 先知道s放在哪里称为SUID/SGID以及如何设置即可，等系统程序章节读完后，再回来看看喔！

- Set UID

当 s 这个标志出现在文件拥有者的 x 权限上时，例如刚刚提到的 /usr/bin/passwd 这个文件的权限状态：“-rwsr-xr-x”，此时就被称为 Set UID，简称为 SUID 的特殊权限。 那么SUID的权限对于一个文件的特殊功能是什么呢？基本上SUID有这样的限制与功能：

- SUID 权限仅对二进制程序（binary program）有效；
- 执行者对于该程序需要具有 x 的可执行权限；
- 本权限仅在执行该程序的过程中有效 （run-time）；
- 执行者将具有该程序拥有者 （owner） 的权限。

讲这么硬的东西你可能对于 SUID 还是没有概念，没关系，我们举个例子来说明好了。 我们的 Linux 系统中，所有帐号的密码都记录在 /etc/shadow 这个文件里面，这个文件的权限为：“---------- 1 root root”，意思是这个文件仅有root可读且仅有root可以强制写入而已。 既然

这个文件仅有 root 可以修改，那么鸟哥的 dmtsai 这个一般帐号使用者能否自行修改自己的密码呢？ 你可以使用你自己的帐号输入“passwd”这个指令来看看，嘿嘿！一般使用者当然可以修改自己的密码了！

唔！有没有冲突啊！明明 /etc/shadow 就不能让 dmtsai 这个一般帐户去存取的，为什么 dmtsai 还能够修改这个文件内的密码呢？ 这就是 SUID 的功能啦！借由上述的功能说明，我们可以知道

1. dmtsai 对于 /usr/bin/passwd 这个程序来说是具有 x 权限的，表示 dmtsai 能执行 passwd；
2. passwd 的拥有者是 root 这个帐号；
3. dmtsai 执行 passwd 的过程中，会“暂时”获得 root 的权限；
4. /etc/shadow 就可以被 dmtsai 所执行的 passwd 所修改。

但如果 dmtsai 使用 cat 去读取 /etc/shadow 时，他能够读取吗？因为 cat 不具有 SUID 的权限，所以 dmtsai 执行 “cat /etc/shadow” 时，是不能读取 /etc/shadow 的。我们用一张示意图来说明如下：

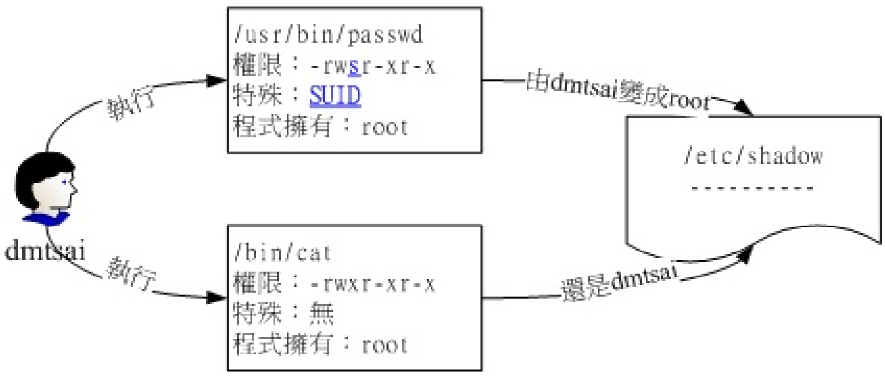


图6.4.1、SUID程序执行的过程示意图

另外，SUID 仅可用在binary program 上， 不能够用在 shell script 上面！这是因为 shell script 只是将很多的 binary 可执行文件叫进来执行而已！所以 SUID 的权限部分，还是得要看 shell script 调用进来的程序的设置， 而不是 shell script 本身。当然，SUID 对于目录也是无效的～这点要特别留意。

- Set GID

当 s 标志在文件拥有者的 x 项目为 SUID，那 s 在群组的 x 时则称为 Set GID, SGID 啰！是这样没错！^_^。 举例来说，你可以用下面的指令来观察到具有 SGID 权限的文件喔：

```
[root@study ~]# ls -l /usr/bin/locate
-rwx--s--x. 1 root slocate 40496 Jun 10  2014 /usr/bin/locate
```

与 SUID 不同的是，SGID 可以针对文件或目录来设置！如果是对文件来说， SGID 有如下的功能：

- SGID 对二进制程序有用；

- 程序执行者对于该程序来说，需具备 x 的权限；
- 执行者在执行的过程中将会获得该程序群组的支持！

举例来说，上面的 /usr/bin/locate 这个程序可以去搜寻 /var/lib/mlocate/mlocate.db 这个文件的内容 （详细说明会在下节讲述），mlocate.db 的权限如下：

```
[root@study ~]# ll /usr/bin/locate /var/lib/mlocate/mlocate.db
-rwx--s--x. 1 root slocate   40496 Jun 10  2014 /usr/bin/locate
-rw-r-----. 1 root slocate 2349055 Jun 15 03:44 /var/lib/mlocate/mlocate.db
```

与 SUID 非常的类似，若我使用 dmtsai 这个帐号去执行 locate 时，那 dmtsai 将会取得 slocate 群组的支持， 因此就能够去读取 mlocate.db 啦！非常有趣吧！

除了 binary program 之外，事实上 SGID 也能够用在目录上，这也是非常常见的一种用途！当一个目录设置了 SGID 的权限后，他将具有如下的功能：

- 使用者若对于此目录具有 r 与 x 的权限时，该使用者能够进入此目录；
- 使用者在此目录下的有效群组（effective group）将会变成该目录的群组；
- 用途：若使用者在此目录下具有 w 的权限（可以新建文件），则使用者所创建的新文件，该新文件的群组与此目录的群组相同。

SGID 对于专案开发来说是非常重要的！因为这涉及群组权限的问题，您可以参考一下本章后续情境仿真的案例，应该就能够对于 SGID 有一些了解的！^_^

- Sticky Bit

这个 Sticky Bit, SBIT 目前只针对目录有效，对于文件已经没有效果了。SBIT 对于目录的作用是：

- 当使用者对于此目录具有 w, x 权限，亦即具有写入的权限时；
- 当使用者在该目录下创建文件或目录时，仅有自己与 root 才有权力删除该文件

换句话说：当甲这个使用者于 A 目录是具有群组或其他人的身份，并且拥有该目录 w 的权限， 这表示“甲使用者对该目录内任何人创建的目录或文件均可进行 "删除/更名/搬移" 等动作。” 不过，如果将 A 目录加上了 SBIT 的权限项目时， 则甲只能够针对自己创建的文件或目录进行删除/更名/移动等动作，而无法删除他人的文件。

举例来说，我们的 /tmp 本身的权限是“drwxrwxrwt”， 在这样的权限内容下，任何人都可以在 /tmp 内新增、修改文件，但仅有该文件/目录创建者与 root 能够删除自己的目录或文件。这个特性也是挺重要的啊！你可以这样做个简单的测试：

1. 以 root 登陆系统，并且进入 /tmp 当中；
2. touch test，并且更改 test 权限成为 777 ；
3. 以一般使用者登陆，并进入 /tmp；
4. 尝试删除 test 这个文件！

由于 SUID/SGID/SBIT 牵涉到程序的概念，因此再次强调，这部份的数据在您读完第十六章关于程序方面的知识后，要再次的回来瞧瞧喔！ 目前，你先有个简单的基础概念就好了！文末的参考数据也建议阅读一番喔！

- SUID/SGID/SBIT 权限设置

前面介绍过 SUID 与 SGID 的功能，那么如何设置文件使成为具有 SUID 与 SGID 的权限呢？这就需要第五章的数字更改权限的方法了！ 现在你应该已经知道数字体态更改权限的方式为“三个数字”的组合， 那么如果在这三个数字之前再加上一个数字的话，最前面的那个数字就代表这几个权限了！

- 4 为 SUID
- 2 为 SGID
- 1 为 SBIT

假设要将一个文件权限改为“-rwsr-xr-x”时，由于 s 在使用者权限中，所以是 SUID ，因此，在原先的 755 之前还要加上 4 ，也就是：“ chmod 4755 filename ”来设置！此外，还有大 S 与大 T 的产生喔！参考下面的范例啦！



Tips 注意：下面的范例只是练习而已，所以鸟哥使用同一个文件来设置，你必须了解 SUID 不是用在目录上，而 SBIT 不是用在文件上的喔！

```
[root@study ~]# cd /tmp
[root@study tmp]# touch test                   &lt;==创建一个测试用空档
[root@study tmp]# chmod 4755 test; ls -l test &lt;==加入具有 SUID 的权限
-rwsr-xr-x 1 root root 0 Jun 16 02:53 test
[root@study tmp]# chmod 6755 test; ls -l test &lt;==加入具有 SUID/SGID 的权限
-rwsr-sr-x 1 root root 0 Jun 16 02:53 test
[root@study tmp]# chmod 1755 test; ls -l test &lt;==加入 SBIT 的功能！
-rwxr-xr-t 1 root root 0 Jun 16 02:53 test
[root@study tmp]# chmod 7666 test; ls -l test &lt;==具有空的 SUID/SGID 权限
-rwSrwSrwT 1 root root 0 Jun 16 02:53 test
```

最后一个例子就要特别小心啦！怎么会出现大写的 S 与 T 呢？不都是小写的吗？ 因为 s 与 t 都是取代 x 这个权限的，但是你有没有发现阿，我们是下达 7666 喔！也就是说， user, group 以及 others 都没有 x 这个可执行的标志（ 因为 666 嘛 ），所以，这个 S, T 代表的就是“空的”啦！怎么说？ SUID 是表示“该文件在执行的时候，具有文件拥有者的权限”，但是文件 拥有者都无法执行了，哪里来的权限给其他人使用？当然就是空的啦！ ^_^

而除了数字法之外，你也可以通过符号法来处理喔！其中 SUID 为 u+s ，而 SGID 为 g+s ，SBIT 则是 o+t 啰！来看看如下的范例：

```
# 设置权限成为 -rws--x--x 的模样：
[root@study tmp]# chmod u=rwxs,go=x test; ls -l test
-rws--x--x 1 root root 0 Jun 16 02:53 test

# 承上，加上 SGID 与 SBIT 在上述的文件权限中！
[root@study tmp]# chmod g+s,o+t test; ls -l test
-rws--s--t 1 root root 0 Jun 16 02:53 test
```

6.4.4 观察文件类型：file

如果你想要知道某个文件的基本数据，例如是属于 ASCII 或者是 data 文件，或者是 binary ，且其中有没有使用到动态函数库 （share library） 等等的信息，就可以利用 file 这个指令来检阅喔！举例来说：

```
[root@study ~]# file ~/.bashrc
/root/.bashrc: ASCII text  &lt;==告诉我们是 ASCII 的纯文本文件啊！
[root@study ~]# file /usr/bin/passwd
/usr/bin/passwd: setuid ELF 64-bit LSB shared object, x86-64, version 1 （SYSV）, dynamica
linked （uses shared libs）, for GNU/Linux 2.6.32,
BuildID[sha1]=0xbf35571e607e317bf107b9bcf65199988d0ed5ab, stripped
# 可执行文件的数据可就多的不得了！包括这个文件的 suid 权限、相容于 Intel x86-64 等级的硬件平台
# 使用的是 Linux 核心 2.6.32 的动态函数库链接等等。
[root@study ~]# file /var/lib/mlocate/mlocate.db
/var/lib/mlocate/mlocate.db: data  &lt;== 这是 data 文件！
```

通过这个指令，我们可以简单的先判断这个文件的格式为何喔！包括未来你也可以用来判断使用 tar 包裹时，该 tarball 文件是使用哪一种压缩功能哩！

6.5 指令与文件的搜寻

文件的搜寻可就厉害了！因为我们常常需要知道那个文件放在哪里，才能够对该文件进行一些修改或维护等动作。 有些时候某些软件配置文件的文件名是不变的，但是各 distribution 放置的目录则不同。 此时就得要利用一些搜寻指令将该配置文件的完整文件名捉出来，这样才能修改嘛！您说是吧！^_^

6.5.1 指令文件名的搜寻

我们知道在终端机模式当中，连续输入两次[tab]按键就能够知道使用者有多少指令可以下达。 那你知不知道这些指令的完整文件名放在哪里？举例来说，ls 这个常用的指令放在哪里呢？ 就通过 which 或 type 来找寻吧！

- which （寻找“可执行文件”）

```
[root@study ~]# which [-a] command
选项或参数：
-a ：将所有由 PATH 目录中可以找到的指令均列出，而不止第一个被找到的指令名称

范例一：搜寻 ifconfig 这个指令的完整文件名
[root@study ~]# which ifconfig
/sbin/ifconfig

范例二：用 which 去找出 which 的文件名为何？
[root@study ~]# which which
alias which='alias &#124; /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
        /bin/alias
        /usr/bin/which
# 竟然会有两个 which ，其中一个是 alias 这玩意儿呢！那是啥？
# 那就是所谓的“命令别名”，意思是输入 which 会等于后面接的那串指令啦！
# 更多的数据我们会在 bash 章节中再来谈的！

范例三：请找出 history 这个指令的完整文件名
[root@study ~]# which history
/usr/bin/which: no history in （/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin:/root/bin）

[root@study ~]# history --help
-bash: history: --: invalid option
history: usage: history [-c] [-d offset] [n] or history -anrw [filename] or history -ps a
# 瞎密？怎么可能没有 history ，我明明就能够用 root 执行 history 的啊！
```

这个指令是根据“PATH”这个环境变量所规范的路径，去搜寻“可执行文件”的文件名～ 所以，重点是找出“可执行文件”而已！且 which 后面接的是“完整文件名”喔！若加上 -a 选项，则可以列出所有的可以找到的同名可执行文件，而非仅显示第一个而已！

最后一个范例最有趣，怎么 history 这个常用的指令竟然找不到啊！为什么呢？这是因为 history 是“bash 内置的指令”啦！ 但是 which 默认是找 PATH 内所规范的目录，所以当然一定找不到的啊（有 bash 就有 history！）！那怎么办？没关系！我们可以通过 type 这个指令喔！ 关于 type 的用法我们将在第十章的 bash 再来谈！

6.5.2 文件文件名的搜寻

再来谈一谈怎么搜寻文件吧！在 Linux 下面也有相当优异的搜寻指令呦！通常 find 不很常用的！因为速度慢之外， 也很操硬盘！一般我们都是先使用 whereis 或者是 locate 来检查，如果真的找不到了，才以 find 来搜寻呦！ 为什么呢？因为 whereis 只找系统中某些特定目录下面的文件而已，locate 则是利用数据库来搜寻文件名，当然两者就相当的快速， 并且没有实际的搜寻硬盘内的文件系统状态，比较省时间啦！

- whereis （由一些特定的目录中寻找文件文件名）

```
[root@study ~]# whereis [-bmsu] 文件或目录名
选项与参数：
-l    :可以列出 whereis 会去查询的几个主要目录而已
-b    :只找 binary 格式的文件
-m    :只找在说明文档 manual 路径下的文件
-s    :只找 source 来源文件
-u    :搜寻不在上述三个项目当中的其他特殊文件

范例一：请找出 ifconfig 这个文件名
[root@study ~]# whereis ifconfig
ifconfig: /sbin/ifconfig /usr/share/man/man8/ifconfig.8.gz

范例二：只找出跟 passwd 有关的“说明文档”文件名（man page）
[root@study ~]# whereis passwd     # 全部的文件名通通列出来！
passwd: /usr/bin/passwd /etc/passwd /usr/share/man/man1/passwd.1.gz /usr/share/man/man5/p
[root@study ~]# whereis -m passwd  # 只有在 man 里面的文件名才抓出来！
passwd: /usr/share/man/man1/passwd.1.gz /usr/share/man/man5/passwd.5.gz
```

等一下我们会提到 find 这个搜寻指令， find 是很强大的搜寻指令，但时间花用的很大！（因为 find 是直接搜寻硬盘，为如果你的硬盘比较老旧的话，嘿嘿！有的等！） 这个时候 whereis 就相当的好用了！另外， whereis 可以加入选项来找寻相关的数据，例如，如果你是要找可执行文件 （binary） 那么加上 -b 就可以啦！ 如果不加任何选项的话，那么就将所有的数据列出来啰！

那么 whereis 到底是使用什么咚咚呢？为何搜寻的速度会比 find 快这么多？ 其实那也没有什么，只是因为 whereis 只找几个特定的目录而已～并没有全系统去查询之故。所以说， whereis 主要是针对 /bin /sbin 下面的可执行文件， 以及 /usr/share/man 下面的 man page 文件，跟几个比较特定的目录来处理而已。所以速度当然快的多！不过，就有某些文件是你找不到的啦！ 想要知道 whereis 到底查了多少目录？可以使用 whereis -l 来确认一下即可！

- locate / updatedb

```
[root@study ~]# locate [-ir] keyword
选项与参数：
-i  ：忽略大小写的差异；
-c  ：不输出文件名，仅计算找到的文件数量
-l  ：仅输出几行的意思，例如输出五行则是 -l 5
-S  ：输出 locate 所使用的数据库文件的相关信息，包括该数据库纪录的文件/目录数量等
-r  ：后面可接正则表达式的显示方式

范例一：找出系统中所有与 passwd 相关的文件名，且只列出 5 个
[root@study ~]# locate -l 5 passwd
/etc/passwd
/etc/passwd-
/etc/pam.d/passwd
/etc/security/opasswd
/usr/bin/gpasswd

范例二：列出 locate 查询所使用的数据库文件之文件名与各数据数量
[root@study ~]# locate -S
Database /var/lib/mlocate/mlocate.db:
        8,086 directories     # 总纪录目录数
        109,605 files         # 总纪录文件数
        5,190,295 Bytes in file names
        2,349,150 Bytes used to store database
```

这个 locate 的使用更简单，直接在后面输入“文件的部分名称”后，就能够得到结果。 举上面的例子来说，我输入 locate passwd ，那么在完整文件名 （包含路径名称） 当中，只要有 passwd 在其中， 就会被显示出来的！这也是个很方便好用的指令，如果你忘记某个文件的完整文件名时～～

但是，这个东西还是有使用上的限制呦！为什么呢？你会发现使用 locate 来寻找数据的时候特别的快， 这是因为 locate 寻找的数据是由“已创建的数据库 /var/lib/mlocate/” 里面的数据所搜寻到的，所以不用直接在去硬盘当中存取数据，呵呵！当然是很快速啰！

那么有什么限制呢？就是因为他是经由数据库来搜寻的，而数据库的创建默认是在每天执行一次 （每个 distribution 都不同，CentOS 7.x 是每天更新数据库一次！） ，所以当你新创建起来的文件， 却还在数据库更新之前搜寻该文件，那么 locate 会告诉你“找不到！”呵呵！因为必须要更新数据库呀！

那能否手动更新数据库哪？当然可以啊！更新 locate 数据库的方法非常简单，直接输入“ updatedb ”就可以了！ updatedb 指令会去读取 /etc/updatedb.conf 这个配置文件的设置，然后再去硬盘里面进行搜寻文件名的动作， 最后就更新整个数据库文件啰！因为 updatedb 会去搜寻硬盘，所以当你执行 updatedb 时，可能会等待数分钟的时间喔！

- updatedb：根据 /etc/updatedb.conf 的设置去搜寻系统硬盘内的文件名，并更新 /var/lib/mlocate 内的数据库文件；
- locate：依据 /var/lib/mlocate 内的数据库记载，找出使用者输入的关键字文件名。

- find

```
[root@study ~]# find [PATH] [option] [action]
选项与参数：
1\. 与时间有关的选项：共有 -atime, -ctime 与 -mtime ，以 -mtime 说明
   -mtime  n ：n 为数字，意义为在 n 天之前的“一天之内”被更动过内容的文件；
   -mtime +n ：列出在 n 天之前（不含 n 天本身）被更动过内容的文件文件名；
   -mtime -n ：列出在 n 天之内（含 n 天本身）被更动过内容的文件文件名。
   -newer file ：file 为一个存在的文件，列出比 file 还要新的文件文件名

范例一：将过去系统上面 24 小时内有更动过内容 （mtime） 的文件列出
[root@study ~]# find / -mtime 0
# 那个 0 是重点！0 代表目前的时间，所以，从现在开始到 24 小时前，
# 有变动过内容的文件都会被列出来！那如果是三天前的 24 小时内？
# find / -mtime 3 有变动过的文件都被列出的意思！

范例二：寻找 /etc 下面的文件，如果文件日期比 /etc/passwd 新就列出
[root@study ~]# find /etc -newer /etc/passwd
# -newer 用在分辨两个文件之间的新旧关系是很有用的！
```

时间参数真是挺有意思的！我们现在知道 atime, ctime 与 mtime 的意义，如果你想要找出一天内被更动过的文件名称， 可以使用上述范例一的作法。但如果我想要找出“4天内被更动过的文件文件名”呢？那可以使用“ find /var -mtime -4 ”。那如果是“4天前的那一天”就用“ find /var -mtime 4 ”。有没有加上“+, -”差别很大喔！我们可以用简单的图示来说明一下：

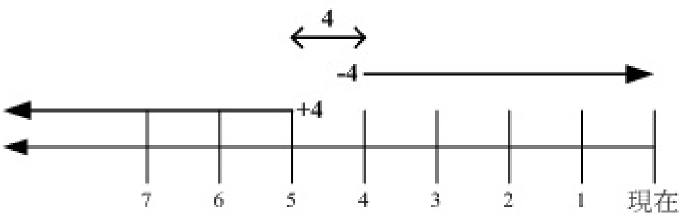


图6.5.1、find 相关的时间参数意义

图中最右边为目前的时间，越往左边则代表越早之前的时间轴啦。由图 6.5.1 我们可以清楚的知道：

- +4代表大于等于5天前的文件名：ex> find /var -mtime +4
- -4代表小于等于4天内的文件文件名：ex> find /var -mtime -4
- 4则是代表4-5那一天的文件文件名：ex> find /var -mtime 4

非常有趣吧！你可以在 /var/ 目录下搜寻一下，感受一下输出文件的差异喔！再来看看其他 find 的用法吧！

```
选项与参数：
2\. 与使用者或群组名称有关的参数：
   -uid n  ：n 为数字，这个数字是使用者的帐号 ID，亦即 UID ，这个 UID 是记录在
            /etc/passwd 里面与帐号名称对应的数字。这方面我们会在第四篇介绍。
   -gid n  ：n 为数字，这个数字是群组名称的 ID，亦即 GID，这个 GID 记录在
            /etc/group，相关的介绍我们会第四篇说明～
   -user name ：name 为使用者帐号名称喔！例如 dmtsai
   -group name：name 为群组名称喔，例如 users ；
   -nouser    ：寻找文件的拥有者不存在 /etc/passwd 的人！
   -nogroup   ：寻找文件的拥有群组不存在于 /etc/group 的文件！
               当你自行安装软件时，很可能该软件的属性当中并没有文件拥有者，
               这是可能的！在这个时候，就可以使用 -nouser 与 -nogroup 搜寻。

范例三：搜寻 /home 下面属于 dmtsai 的文件
[root@study ~]# find /home -user dmtsai
# 这个东西也很有用的～当我们要找出任何一个使用者在系统当中的所有文件时，
# 就可以利用这个指令将属于某个使用者的所有文件都找出来喔！

范例四：搜寻系统中不属于任何人的文件
[root@study ~]# find / -nouser
# 通过这个指令，可以轻易的就找出那些不太正常的文件。如果有找到不属于系统任何人的文件时，
# 不要太紧张，那有时候是正常的～尤其是你曾经以源代码自行编译软件时。
```

如果你想要找出某个使用者在系统下面创建了啥咚咚，使用上述的选项与参数，就能够找出来啦！ 至于那个 -nouser 或 -nogroup 的选项功能中，除了你自行由网络上面下载文件时会发生之外， 如果你将系统里面某个帐号删除了，但是该帐号已经在系统内创建很多文件时，就可能会发生无主孤魂的文件存在！ 此时你就得使用这个 -nouser 来找出该类型的文件啰！

```
选项与参数：
3\. 与文件权限及名称有关的参数：
   -name filename：搜寻文件名称为 filename 的文件；
   -size [+-]SIZE：搜寻比 SIZE 还要大（+）或小（-）的文件。这个 SIZE 的规格有：
                  c: 代表 Byte， k: 代表 1024Bytes。所以，要找比 50KB
                  还要大的文件，就是“ -size +50k ”
   -type TYPE    ：搜寻文件的类型为 TYPE 的，类型主要有：一般正规文件 （f）, 设备文件 （b, c）,
                  目录 （d）, 链接文件 （l）, socket （s）, 及 FIFO （p） 等属性。
   -perm mode  ：搜寻文件权限“刚好等于” mode 的文件，这个 mode 为类似 chmod
                的属性值，举例来说， -rwsr-xr-x 的属性为 4755 ！
   -perm -mode ：搜寻文件权限“必须要全部囊括 mode 的权限”的文件，举例来说，
                我们要搜寻 -rwxr--r-- ，亦即 0744 的文件，使用 -perm -0744，
                当一个文件的权限为 -rwsr-xr-x ，亦即 4755 时，也会被列出来，
                因为 -rwsr-xr-x 的属性已经囊括了 -rwxr--r-- 的属性了。
   -perm /mode ：搜寻文件权限“包含任一 mode 的权限”的文件，举例来说，我们搜寻
                -rwxr-xr-x ，亦即 -perm /755 时，但一个文件属性为 -rw-------
                也会被列出来，因为他有 -rw.... 的属性存在！

范例五：找出文件名为 passwd 这个文件
[root@study ~]# find / -name passwd

范例五-1：找出文件名包含了 passwd 这个关键字的文件
[root@study ~]# find / -name "*passwd*"
# 利用这个 -name 可以搜寻文件名啊！默认是完整文件名，如果想要找关键字，
# 可以使用类似 * 的任意字符来处理

范例六：找出 /run 目录下，文件类型为 Socket 的文件名有哪些？
[root@study ~]# find /run -type s
# 这个 -type 的属性也很有帮助喔！尤其是要找出那些怪异的文件，
# 例如 socket 与 FIFO 文件，可以用 find /run -type p 或 -type s 来找！

范例七：搜寻文件当中含有 SGID 或 SUID 或 SBIT 的属性
[root@study ~]# find / -perm /7000
# 所谓的 7000 就是 ---s--s--t ，那么只要含有 s 或 t 的就列出，所以当然要使用 /7000，
# 使用 -7000 表示要同时含有 ---s--s--t 的所有三个权限。而只需要任意一个，就是 /7000 ～瞭乎？
```

上述范例中比较有趣的就属 -perm 这个选项啦！他的重点在找出特殊权限的文件啰！ 我们知道 SUID 与 SGID 都可以设置在二进制程序上，假设我想要找出来 /usr/bin, /usr/sbin 这两个目录下， 只要具有 SUID 或 SGID 就列出来该文件，你可以这样做：

```
[root@study ~]# find /usr/bin /usr/sbin -perm /6000
```

因为 SUID 是 4 分，SGID 2 分，总共为 6 分，因此可用 /6000 来处理这个权限！ 至于 find 后面可以接多个目录来进行搜寻！另外， find 本来就会搜寻次目录，这个特色也要特别注意喔！ 最后，我们再来看一下 find 还有什么特殊功能吧！

```
选项与参数：
4\. 额外可进行的动作：
   -exec command ：command 为其他指令，-exec 后面可再接额外的指令来处理搜寻到的结果。
   -print        ：将结果打印到屏幕上，这个动作是默认动作！

范例八：将上个范例找到的文件使用 ls -l 列出来～
[root@study ~]# find /usr/bin /usr/sbin -perm /7000 -exec ls -l {} \;
# 注意到，那个 -exec 后面的 ls -l 就是额外的指令，指令不支持命令别名，
# 所以仅能使用 ls -l 不可以使用 ll 喔！注意注意！

范例九：找出系统中，大于 1MB 的文件
[root@study ~]# find / -size +1M
```

find 的特殊功能就是能够进行额外的动作（action）。我们将范例八的例子以图解来说明如下：

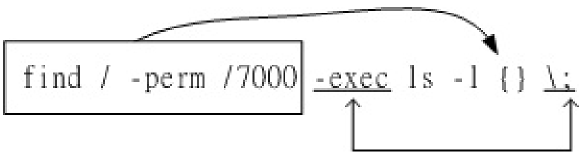


图6.5.2、find 相关的额外动作

该范例中特殊的地方有 {} 以及 \; 还有 -exec 这个关键字，这些东西的意义为：

- {} 代表的是“由 find 找到的内容”，如上图所示，find 的结果会被放置到 {} 位置中；
- -exec 一直到 \; 是关键字，代表 find 额外动作的开始 （-exec） 到结束 （\;） ，在这中间的就是 find 指令内的额外动作。 在本例中就是“ ls -l {} ”啰！
- 因为“ ; ”在 bash 环境下是有特殊意义的，因此利用反斜线来跳脱。

通过图 6.5.2 你应该就比较容易了解 -exec 到 \; 之间的意义了吧！

如果你要找的文件是具有特殊属性的，例如 SUID 、文件拥有者、文件大小等等， 那么利用 locate 是没有办法达成你的搜寻的！此时 find 就显的很重要啦！ 另外，find 还可以利用万用字符来找寻文件名呢！举例来说，你想要找出 /etc 下面文件名包含 httpd 的文件， 那么你就可以这样做：

```
[root@study ~]# find /etc -name '*httpd*'
```

不但可以指定搜寻的目录（连同次目录），并且可以利用额外的选项与参数来找到最正确的文件名！真是好好用！ 不过由于 find 在寻找数据的时后相当的操硬盘！所以没事情不要使用 find 啦！有更棒的指令可以取代呦！那就是上面提到的 whereis 与 locate 啰！

6.6 极重要的复习！权限与指令间的关系

我们知道权限对于使用者帐号来说是非常重要的，因为他可以限制使用者能不能读取/创建/删除/修改文件或目录！ 在这一章我们介绍了很多文件系统的管理指令，第五章则介绍了很多文件权限的意义。在这个小节当中， 我们就将这两者结合起来，说明一下什么指令在什么样的权限下才能够运行吧！^_^

一、让使用者能进入某目录成为“可工作目录”的基本权限为何：

- 可使用的指令：例如 cd 等变换工作目录的指令；
- 目录所需权限：使用者对这个目录至少需要具有 x 的权限
- 额外需求：如果使用者想要在这个目录内利用 ls 查阅文件名，则使用者对此目录还需要 r 的权限。

二、使用者在某个目录内读取一个文件的基本权限为何？

- 可使用的指令：例如本章谈到的 cat, more, less等等
- 目录所需权限：使用者对这个目录至少需要具有 x 权限；
- 文件所需权限：使用者对文件至少需要具有 r 的权限才行！

三、让使用者可以修改一个文件的基本权限为何？

- 可使用的指令：例如 nano 或未来要介绍的 vi 编辑器等；
- 目录所需权限：使用者在该文件所在的目录至少要有 x 权限；
- 文件所需权限：使用者对该文件至少要有 r, w 权限

四、让一个使用者可以创建一个文件的基本权限为何？

- 目录所需权限：使用者在该目录要具有 w,x 的权限，重点在 w 啦！

五、让使用者进入某目录并执行该目录下的某个指令之基本权限为何？

- 目录所需权限：使用者在该目录至少要有 x 的权限；
- 文件所需权限：使用者在该文件至少需要有 x 的权限

例题：让一个使用者 dmtsai 能够进行“cp /dir1/file1 /dir2”的指令时，请说明 dir1, file1, dir2 的最小所需权限为何？答：执行 cp 时， dmtsai 要“能够读取来源文件，并且写入目标文件！”所以应参考上述第二点与第四点的说明！ 因此各文件/目录的最小权限应该是：

- dir1 ：至少需要有 x 权限；
- file1：至少需要有 r 权限；
- dir2 ：至少需要有 w, x 权限。

例题：有一个文件全名为 /home/student/www/index.html ，各相关文件/目录的权限如下：

```
drwxr-xr-x 23 root    root    4096 Sep 22 12:09 /
drwxr-xr-x  6 root    root    4096 Sep 29 02:21 /home
drwx------  6 student student 4096 Sep 29 02:23 /home/student
drwxr-xr-x  6 student student 4096 Sep 29 02:24 /home/student/www
-rwxr--r--  6 student student  369 Sep 29 02:27 /home/student/www/index.html
```

请问 vbird 这个帐号（不属于student群组）能否读取 index.html 这个文件呢？答：虽然 www 与 index.html 是可以让 vbird 读取的权限，但是因为目录结构是由根目录一层一层读取的，因此 vbird 可进入 /home 但是却不可进入 /home/student/ ，既然连进入 /home/student 都不许了， 当然就读不到 index.html 了！所以答案是“vbird不会读取到 index.html 的内容”喔！

那要如何修改权限呢？其实只要将 /home/student 的权限修改为最小 711 ，或者直接给予 755 就可以啰！ 这可是很重要的概念喔！

6.7 重点回顾

- 绝对路径："一定由根目录 / 写起"；相对路径："不由 / 写起，而是由相对当前目录写起"
- 特殊目录有：., .., -, ~, ~account需要注意；
- 与目录相关的指令有：cd, mkdir, rmdir, pwd 等重要指令；
- rmdir 仅能删除空目录，要删除非空目录需使用" rm -r "指令；
- 使用者能使用的指令是依据 PATH 变量所规定的目录去搜寻的；
- ls 可以检视文件的属性，尤其 -d, -a, -l 等选项特别重要！
- 文件的复制、删除、移动可以分别使用：cp, rm , mv等指令来操作；
- 检查文件的内容（读档）可使用的指令包括有：cat, tac, nl, more, less, head, tail, od 等
- cat -n 与 nl 均可显示行号，但默认的情况下，空白行会不会编号并不相同；
- touch 的目的在修改文件的时间参数，但亦可用来创建空文件；
- 一个文件记录的时间参数有三种，分别是 access time（atime）, status time （ctime）, modification time（mtime），ls 默认显示的是 mtime。
- 除了传统的rwx权限之外，在Ext2/Ext3/Ext4/xfs文件系统中，还可以使用chattr与lsattr设置及观察隐藏属性。 常见的包括只能新增数据的 +a 与完全不能更动文件的 +i 属性。
- 新建文件/目录时，新文件的默认权限使用 umask 来规范。默认目录完全权限为drwxrwxrwx， 文件则为-rw-rw-rw-。
- 文件具有SUID的特殊权限时，代表当使用者执行此一binary程序时，在执行过程中使用者会暂时具有程序拥有者的权限
- 目录具有SGID的特殊权限时，代表使用者在这个目录下面新建的文件之群组都会与该目录的群组名称相同。
- 目录具有SBIT的特殊权限时，代表在该目录下使用者创建的文件只有自己与root能够删除！
- 观察文件的类型可以使用 file 指令来观察；
- 搜寻指令的完整文件名可用 which 或 type ，这两个指令都是通过 PATH 变量来搜寻文件名；
- 搜寻文件的完整文件名可以使用 whereis 找特定目录或 locate 到数据库去搜寻，而不实际搜寻文件系统；
- 利用 find 可以加入许多选项来直接查询文件系统，以获得自己想要知道的文件名。

6.8 本章习题：

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ）

情境仿真题一：假设系统中有两个帐号，分别是 alex 与 arod ，这两个人除了自己群组之外还共同支持一个名为 project 的群组。假设这两个用户需要共同拥有 /srv/ahome/ 目录的开发权，且该目录不许其他人进入查阅。 请问该目录的权限设置应为何？请先以传统权限说明，再以 SGID 的功能解析。

- 目标：了解到为何专案开发时，目录最好需要设置 SGID 的权限！
- 前提：多个帐号支持同一群组，且共同拥有目录的使用权！
- 需求：需要使用 root 的身份来进行 chmod, chgrp 等帮用户设置好他们的开发环境才行！这也是管理员的重要任务之一！

首先我们得要先制作出这两个帐号的相关数据，帐号/群组的管理在后续我们会介绍， 您这里先照着下面的指令来制作即可：

```
[root@study ~]# groupadd project        &lt;==增加新的群组
[root@study ~]# useradd -G project alex &lt;==创建 alex 帐号，且支持 project
[root@study ~]# useradd -G project arod &lt;==创建 arod 帐号，且支持 project
[root@study ~]# id alex                 &lt;==查阅 alex 帐号的属性
uid=1001（alex） gid=1002（alex） groups=1002（alex）,1001（project） &lt;==确实有支持！
[root@study ~]# id arod
uid=1002（arod） gid=1003（arod） groups=1003（arod）,1001（project） &lt;==确实有支持！
```

然后开始来解决我们所需要的环境吧！

1. 首先创建所需要开发的专案目录：

    ```
    [root@study ~]# mkdir /srv/ahome
    [root@study ~]# ll -d /srv/ahome
    drwxr-xr-x. 2 root root 6 Jun 17 00:22 /srv/ahome
    ```

2. 从上面的输出结果可发现 alex 与 arod 都不能在该目录内创建文件，因此需要进行权限与属性的修改。 由于其他人均不可进入此目录，因此该目录的群组应为project，权限应为770才合理。

    ```
    [root@study ~]# chgrp project /srv/ahome
    [root@study ~]# chmod 770 /srv/ahome
    [root@study ~]# ll -d /srv/ahome
    drwxrwx---. 2 root project 6 Jun 17 00:22 /srv/ahome
    # 从上面的权限结果来看，由于 alex/arod 均支持 project，因此似乎没问题了！
    ```

3. 实际分别以两个使用者来测试看看，情况会是如何？先用 alex 创建文件，然后用 arod 去处理看看。

```
[root@study ~]# su - alex       &lt;==先切换身份成为 alex 来处理
[alex@www ~]$ cd /srv/ahome   &lt;==切换到群组的工作目录去
[alex@www ahome]$ touch abcd  &lt;==创建一个空的文件出来！
[alex@www ahome]$ exit        &lt;==离开 alex 的身份

[root@study ~]# su - arod
[arod@www ~]$ cd /srv/ahome
[arod@www ahome]$ ll abcd
-rw-rw-r--. 1 alex alex 0 Jun 17 00:23 abcd
# 仔细看一下上面的文件，由于群组是 alex ，arod并不支持！
# 因此对于 abcd 这个文件来说， arod 应该只是其他人，只有 r 的权限而已啊！
[arod@www ahome]$ exit
```

由上面的结果我们可以知道，若单纯使用传统的 rwx 而已，则对刚刚 alex 创建的 abcd 这个文件来说， arod 可以删除他，但是却不能编辑他！这不是我们要的样子啊！赶紧来重新规划一下。

4. 加入 SGID 的权限在里面，并进行测试看看：

```
[root@study ~]# chmod 2770 /srv/ahome
[root@study ~]# ll -d /srv/ahome
drwxrws---. 2 root project 17 Jun 17 00:23 /srv/ahome

测试：使用 alex 去创建一个文件，并且查阅文件权限看看：
[root@study ~]# su - alex
[alex@www ~]$ cd /srv/ahome
[alex@www ahome]$ touch 1234
[alex@www ahome]$ ll 1234
-rw-rw-r--. 1 alex project 0 Jun 17 00:25 1234
# 没错！这才是我们要的样子！现在 alex, arod 创建的新文件所属群组都是 project，
# 由于两人均属于此群组，加上 umask 都是 002，这样两人才可以互相修改对方的文件！
```

所以最终的结果显示，此目录的权限最好是“2770”，所属文件拥有者属于root即可，至于群组必须要为两人共同支持的project 这个群组才行！

简答题部分：

- 什么是绝对路径与相对路径绝对路径的写法为由 / 开始写，至于相对路径则不由 / 开始写！此外，相对路径为相对于目前工作目录的路径！
- 如何更改一个目录的名称？例如由 /home/test 变为 /home/test2mv /home/test /home/test2
- PATH 这个环境变量的意义？这个是用来指定可执行文件执行的时候，指令搜寻的目录路径。
- umask 有什么用处与优点？umask 可以拿掉一些权限，因此，适当的定义 umask 有助于系统的安全， 因为他可以用来创建默认的目录或文件的权限。
- 当一个使用者的 umask 分别为 033 与 044 他所创建的文件与目录的权限为何？在 umask 为 033 时，则默认是拿掉 group 与 other 的 w（2）x（1） 权限，因此权限就成

为“文件 -rw-r--r-- ， 目录 drwxr--r-- ”而当 umask 044 时，则拿掉 r 的属性，因此就成为“文件 -rw--w--w-，目录 drwx-wx-wx”

- 什么是 SUID ？当一个指令具有 SUID 的功能时，则：
    - SUID 权限仅对二进制程序（binary program）有效；
    - 执行者对于该程序需要具有 x 的可执行权限；
    - 本权限仅在执行该程序的过程中有效 （run-time）；
    - 执行者将具有该程序拥有者 （owner） 的权限。
- 当我要查询 /usr/bin/passwd 这个文件的一些属性时（1）传统权限；（2）文件类型与（3）文件的隐藏属性，可以使用什么指令来查询？ls -al file lsattr
- 尝试用 find 找出目前 linux 系统中，所有具有 SUID 的文件有哪些？find / -perm +4000 -print
- 找出 /etc 下面，文件大小介于 50K 到 60K 之间的文件，并且将权限完整的列出 （ls -l）：find /etc -size +50k -a -size -60k -exec ls -l {} \; 注意到 -a ，那个 -a 是 and 的意思，为符合两者才算成功
- 找出 /etc 下面，文件大小大于 50K 且文件所属人不是 root 的文件名，且将权限完整的列出 （ls -l）；find /etc -size +50k -a ! -user root -exec ls -ld {} \; find /etc -size +50k -a ! -user root -type f -exec ls -l {} \; 上面两式均可！注意到！，那个！代表的是反向选择，亦即“不是后面的项目”之意！
- 找出 /etc 下面，容量大于 1500K 以及容量等于 0 的文件：find /etc -size +1500k -o -size 0 相对于 -a ，那个 -o 就是或 （or） 的意思啰！

6.9 参考资料与延伸阅读

- 小洲大大回答 SUID/SGID 的一篇讨论： http://phorum.vbird.org/viewtopic.php?t=20256

2002/06/26：第一次完成 2003/02/06：重新编排与加入 FAQ 2003/02/07：加入 basename 与 dirname 的说明 2004/03/15：将链接文件的内容移动至下一章节：Linux 磁盘与硬件管理 2005/07/19：将旧的文章移动到 这里 了。 2005/07/20：呼呼！好不容易啊～在被台风尾扫到的七月份，终于写完这个咚咚～ 2005/07/21：在 find 部分，多增加了范例九，以及关于利用文件大小 （size） 搜寻的功能。 2005/07/25：在 SUID/SGID/SBIT 部分，依据 netman 与 小州 兄的建议，修改了部分的叙述！ 2006/04/09：在 rmdir 的范例内，少了一个 -p 的参数！ 2006/06/15：经由讨论区网友 dm421 的通知，发现 chattr 的部分关于 d 写错了，已订正。 2006/08/22：增加 rm 的一些简单的说明！尤其是“ rm ./-aaa- ”的删除方法！ 2008/09/23：将针对FC4版写的数据移到此处 2008/09/29：加入权限与指令的关系一节，并新增情境仿真题目喔！大家帮忙除错一下！ 2009/08/18：加入符号法的方式来处理 SUID/SGID/SBIT 啰！ 2009/08/26：感谢网友告知习题部分，找出 /etc 下面容量大于 50k 的那题，应使用 -type f 或 ls -ld 来避免目录内重复显示！ 2015/06/04：将旧的基于 CentOS 5 的文章移动到此处。 2015/06/24：感谢网友“学习日记博客”的告知，whereis 以前一直写错了！这次给它订正一下！感谢！ 2015/08/25：感谢网友“学习日记博客”的告知，cp 的参数内， -a 不仅代表 -pdr ！因为有 SELinux 的影响的关系！

第七章、Linux 磁盘与文件系统管理

最近更新日期：20//

系统管理员很重要的任务之一就是管理好自己的磁盘文件系统，每个分区不可太大也不能太小， 太大会造成磁盘容量的浪费，太小则会产生文件无法储存的困扰。此外，我们在前面几章谈到的文件权限与属性中， 这些权限与属性分别记录在文件系统的哪个区块内？这就得要谈到 filesystem 中的 inode 与 block 了。同时，为了虚拟化与大容量磁盘， 现在的 CentOS 7 默认使用大容量性能较佳的 xfs 当默认文件系统了！这也得了解一下。 在本章我们的重点在于如何制作文件系统，包括分区、格式化与挂载等，是很重要的一个章节喔！

7.1 认识 Linux 文件系统

Linux 最传统的磁盘文件系统 （filesystem） 使用的是 EXT2 这个啦！所以要了解 Linux 的文件系统就得要由认识 EXT2 开始！ 而文件系统是创建在磁盘上面的，因此我们得了解磁盘的物理组成才行。磁盘物理组成的部分我们在第零章谈过了，至于磁盘分区则在第二章谈过了，所以下面只会很快的复习这两部份。 重点在于 inode, block 还有 superblock 等文件系统的基本部分喔！

7.1.1 磁盘组成与分区的复习

由于各项磁盘的物理组成我们在第零章里面就介绍过， 同时第二章也谈过分区的概念了，所以这个小节我们就拿之前的重点出来介绍就好了！ 详细的信息请您回去那两章自行复习喔！^_^。好了，首先说明一下磁盘的物理组成，整颗磁盘的组成主要有：

- 圆形的盘片（主要记录数据的部分）；
- 机械手臂，与在机械手臂上的磁头（可读写盘片上的数据）；
- 主轴马达，可以转动盘片，让机械手臂的磁头在盘片上读写数据。

从上面我们知道数据储存与读取的重点在于盘片，而盘片上的物理组成则为（假设此磁盘为单碟片， 盘片图示请参考第二章图2.2.1的示意）：

- 扇区（Sector）为最小的物理储存单位，且依据磁盘设计的不同，目前主要有 512Bytes 与 4K 两种格式；
- 将扇区组成一个圆，那就是柱面（Cylinder）；
- 早期的分区主要以柱面为最小分区单位，现在的分区通常使用扇区为最小分区单位（每个扇区都有其号码喔，就好像座位一样）；
- 磁盘分区表主要有两种格式，一种是限制较多的 MBR 分区表，一种是较新且限制较少的 GPT 分区表。
- MBR 分区表中，第一个扇区最重要，里面有：（1）主要开机区（Master boot record, MBR）及分区表（partition table）， 其中 MBR 占有 446 Bytes，而 partition table 则占有 64 Bytes。
- GPT 分区表除了分区数量扩充较多之外，支持的磁盘容量也可以超过 2TB。

至于磁盘的文件名部份，基本上，所有实体磁盘的文件名都已经被仿真成 /dev/sd[a-p] 的格式，第一颗磁盘文件名为 /dev/sda。 而分区的文件名若以第一颗磁盘为例，则为 /dev/sda[1-128] 。除了实体磁盘之外，虚拟机的磁盘通常为 /dev/vd[a-p] 的格式。 若有使用到软件磁盘阵列的话，那还有 /dev/md[0-128] 的磁盘文件名。使用的是 LVM 时，文件名则为 /dev/VGNAME/LVNAME 等格式。 关于软件磁盘阵列与 LVM 我们会在后面继续介绍，这里主要介绍的以实体磁盘及虚拟磁盘为主喔！

- /dev/sd[a-p][1-128]：为实体磁盘的磁盘文件名；

- /dev/vd[a-d][1-128]：为虚拟磁盘的磁盘文件名

复习完物理组成后，来复习一下磁盘分区吧！如前所述，以前磁盘分区最小单位经常是柱面，但 CentOS 7 的分区软件， 已经将最小单位改成扇区了，所以容量大小的分区可以切的更细～此外，由于新的大容量磁盘大多得要使用 GPT 分区表才能够使用全部的容量， 因此过去那个 MBR 的传统磁盘分区表限制就不会存在了。不过，由于还是有小磁盘啊！因此， 你在处理分区的时候，还是得要先查询一下，你的分区是 MBR 的分区？还是 GPT 的分区？在第三章的 CentOS 7 安装中， 鸟哥建议过强制使用 GPT 分区喔！所以本章后续的动作，大多还是以 GPT 为主来介绍喔！旧的 MBR 相关限制回去看看第二章吧！

7.1.2 文件系统特性

我们都知道磁盘分区完毕后还需要进行格式化（format），之后操作系统才能够使用这个文件系统。 为什么需要进行“格式化”呢？这是因为每种操作系统所设置的文件属性/权限并不相同， 为了存放这些文件所需的数据，因此就需要将分区进行格式化，以成为操作系统能够利用的“文件系统格式（filesystem）”。

由此我们也能够知道，每种操作系统能够使用的文件系统并不相同。 举例来说，windows 98 以前的微软操作系统主要利用的文件系统是 FAT （或 FAT16），windows 2000 以后的版本有所谓的 NTFS 文件系统，至于 Linux 的正统文件系统则为 Ext2 （Linux second extended file system, ext2fs）这一个。此外，在默认的情况下，windows 操作系统是不会认识 Linux 的 Ext2 的。

传统的磁盘与文件系统之应用中，一个分区就是只能够被格式化成为一个文件系统，所以我们可以说一个 filesystem 就是一个 partition。但是由于新技术的利用，例如我们常听到的LVM与软件磁盘阵列（software raid）， 这些技术可以将一个分区格式化为多个文件系统（例如LVM），也能够将多个分区合成一个文件系统（LVM, RAID）！ 所以说，目前我们在格式化时已经不再说成针对 partition 来格式化了， 通常我们可以称呼一个可被挂载的数据为一个文件系统而不是一个分区喔！

那么文件系统是如何运行的呢？这与操作系统的文件数据有关。较新的操作系统的文件数据除了文件实际内容外， 通常含有非常多的属性，例如 Linux 操作系统的文件权限（rwx）与文件属性（拥有者、群组、时间参数等）。 文件系统通常会将这两部份的数据分别存放在不同的区块，权限与属性放置到 inode 中，至于实际数据则放置到 data block 区块中。 另外，还有一个超级区块 （superblock） 会记录整个文件系统的整体信息，包括 inode 与 block 的总量、使用量、剩余量等。

每个 inode 与 block 都有编号，至于这三个数据的意义可以简略说明如下：

- superblock：记录此 filesystem 的整体信息，包括inode/block的总量、使用量、剩余量，以及文件系统的格式与相关信息等；
- inode：记录文件的属性，一个文件占用一个inode，同时记录此文件的数据所在的 block 号码；

- block：实际记录文件的内容，若文件太大时，会占用多个 block 。

由于每个 inode 与 block 都有编号，而每个文件都会占用一个 inode ，inode 内则有文件数据放置的 block 号码。 因此，我们可以知道的是，如果能够找到文件的 inode 的话，那么自然就会知道这个文件所放置数据的 block 号码， 当然也就能够读出该文件的实际数据了。这是个比较有效率的作法，因为如此一来我们的磁盘就能够在短时间内读取出全部的数据， 读写的性能比较好啰。

我们将 inode 与 block 区块用图解来说明一下，如下图所示，文件系统先格式化出 inode 与 block 的区块，假设某一个文件的属性与权限数据是放置到 inode 4 号（下图较小方格内），而这个 inode 记录了文件数据的实际放置点为 2, 7, 13, 15 这四个 block 号码，此时我们的操作系统就能够据此来排列磁盘的读取顺序，可以一口气将四个 block 内容读出来！ 那么数据的读取就如同下图中的箭头所指定的模样了。

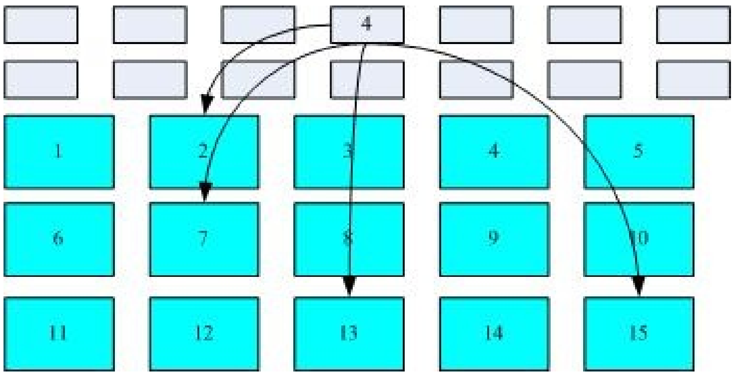


图7.1.1、inode/block 数据存取示意图

这种数据存取的方法我们称为索引式文件系统（indexed allocation）。那有没有其他的惯用文件系统可以比较一下啊？ 有的，那就是我们惯用的U盘（闪存），U盘使用的文件系统一般为 FAT 格式。FAT 这种格式的文件系统并没有 inode 存在，所以 FAT 没有办法将这个文件的所有 block 在一开始就读取出来。每个 block 号码都记录在前一个 block 当中， 他的读取方式有点像下面这样：

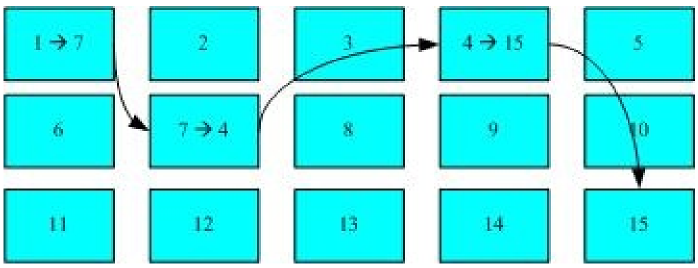


图7.1.2、FAT文件系统数据存取示意图

上图中我们假设文件的数据依序写入1->7->4->15号这四个 block 号码中， 但这个文件系统没有办法一口气就知道四个 block 的号码，他得要一个一个的将 block 读出后，才会知道下一个 block 在何处。 如果同一个文件数据写入的 block 分散的太过厉害时，则我们的磁头将无法在磁盘转一圈就读到所有的数据， 因此磁盘就会多转好几圈才能完整的读取到这个文件的内容！

常常会听到所谓的“磁盘重组”吧？ 需要磁盘重组的原因就是文件写入的 block 太过于离散了，此时文件读取的性能将会变的很差所致。 这个时候可以通过磁盘重组将同一个文件所属的 blocks 汇整在一起，这样数据的读取会比较容易啊！ 想当然尔，FAT 的文件系统需要三不五时的磁盘重组一下，那么 Ext2 是否需要磁盘重整呢？

由于 Ext2 是索引式文件系统，基本上不太需要常常进行磁盘重组的。但是如果文件系统使用太久， 常常删除/编辑/新增文件时，那么还是可能会造成文件数据太过于离散的问题，此时或许会需要进行重整一下的。 不过，老实说，鸟哥倒是没有在 Linux 操作系统上面进行过 Ext2/Ext3 文件系统的磁盘重组说！似乎不太需要啦！^_^

7.1.3 Linux 的 EXT2 文件系统（inode）

在第五章当中我们介绍过 Linux 的文件除了原有的数据内容外，还含有非常多的权限与属性，这些权限与属性是为了保护每个使用者所拥有数据的隐密性。 而前一小节我们知道 filesystem 里面可能含有的 inode/block/superblock 等。为什么要谈这个呢？因为标准的 Linux 文件系统 Ext2 就是使用这种 inode 为基础的文件系统啦！

而如同前一小节所说的，inode 的内容在记录文件的权限与相关属性，至于 block 区块则是在记录文件的实际内容。 而且文件系统一开始就将 inode 与 block 规划好了，除非重新格式化（或者利用 resize2fs 等指令变更文件系统大小），否则 inode 与 block 固定后就不再变动。 但是如果仔细考虑一下，如果我的文件系统高达数百GB时， 那么将所有的 inode 与 block 通通放置在一起将是很不智的决定，因为 inode 与 block 的数量太庞大，不容易管理。

为此之故，因此 Ext2 文件系统在格式化的时候基本上是区分为多个区块群组 （block group） 的，每个区块群组都有独立的 inode/block/superblock 系统。感觉上就好像我们在当兵时，一个营里面有分成数个连，每个连有自己的联络系统， 但最终都向营部回报连上最正确的信息一般！这样分成一群群的比较好管理啦！整个来说，Ext2 格式化后有点像下面这样：

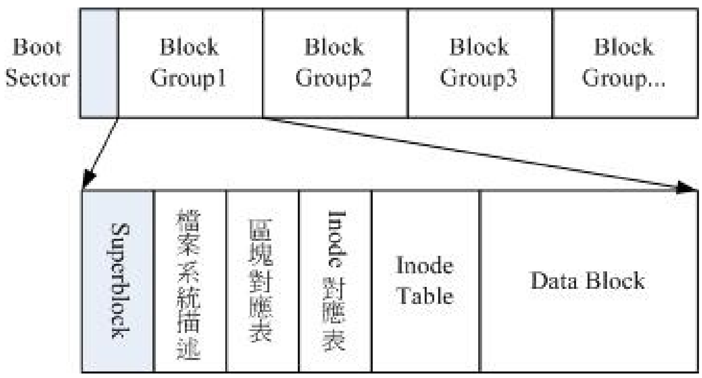


图7.1.3、ext2文件系统示意图 [1]

在整体的规划当中，文件系统最前面有一个开机扇区（boot sector），这个开机扇区可以安装开机管理程序， 这是个非常重要的设计，因为如此一来我们就能够将不同的开机管理程序安装到个别的文件系统最前端，而不用覆盖整颗磁盘唯一的 MBR， 这样也才能够制作出多重开机的环境啊！至于每一个区块群组（block group）的六个主要内容说明如后：

- data block （数据区块）

data block 是用来放置文件内容数据地方，在 Ext2 文件系统中所支持的 block 大小有 1K, 2K 及 4K 三种而已。在格式化时 block 的大小就固定了，且每个 block 都有编号，以方便 inode 的记录啦。 不过要注意的是，由于 block 大小的差异，会导致该文件系统能够支持的最大磁盘容量与最大单一文件大小并不相同。 因为 block 大小而产生的 Ext2 文件系统限制如下：[2]

Block 大小	1KB	2KB	4KB
最大单一文件限制	16GB	256GB	2TB
最大文件系统总容量	2TB	8TB	16TB

你需要注意的是，虽然 Ext2 已经能够支持大于 2GB 以上的单一文件大小，不过某些应用程序依然使用旧的限制， 也就是说，某些程序只能够捉到小于 2GB 以下的文件而已，这就跟文件系统无关了！ 举例来说，鸟哥在环工方面的应用中有一套秀图软件称为PAVE[3]， 这套软件就无法捉到鸟哥在数值模式仿真后产生的大于 2GB 以上的文件！所以后来只能找更新的软件来取代它了！

除此之外 Ext2 文件系统的 block 还有什么限制呢？有的！基本限制如下：

- 原则上，block 的大小与数量在格式化完就不能够再改变了（除非重新格式化）；
- 每个 block 内最多只能够放置一个文件的数据；
- 承上，如果文件大于 block 的大小，则一个文件会占用多个 block 数量；
- 承上，若文件小于 block ，则该 block 的剩余容量就不能够再被使用了（磁盘空间会浪费）。

如上第四点所说，由于每个 block 仅能容纳一个文件的数据而已，因此如果你的文件都非常小，但是你的 block 在格式化时却选用最大的 4K 时，可能会产生一些容量的浪费喔！我们以下面的一个简单例题来算一下空间的浪费吧！

例题：假设你的Ext2文件系统使用 4K block ，而该文件系统中有 10000 个小文件，每个文件大小均为 50Bytes， 请问此时你的磁盘浪费多少容量？答：由于 Ext2 文件系统中一个 block 仅能容纳一个文件，因此每个 block 会浪费“ 4096 - 50 = 4046 （Byte）”， 系统中总共有一万个小文件，所有文件大小为：50 （Bytes） x 10000 = 488.3KBytes，但此时浪费的容量为：“ 4046 （Bytes） x 10000 = 38.6MBytes ”。想一想，不到 1MB 的总文件大小却浪费将近 40MB 的容量，且文件越多将造成越多的磁盘容量浪费。

什么情况会产生上述的状况呢？例如 BBS 网站的数据啦！如果 BBS 上面的数据使用的是纯文本来记载每篇留言， 而留言内容如果都写上“如题”时，想一想，是否就会产生很多小文件了呢？

好，既然大的 block 可能会产生较严重的磁盘容量浪费，那么我们是否就将 block 大小订为 1K 即可？ 这也不妥，因为如果 block 较小的话，那么大型文件将会占用数量更多的 block ，而 inode 也要记录更多的 block 号码，此时将可能导致文件系统不良的读写性能。

所以我们可以说，在您进行文件系统的格式化之前，请先想好该文件系统预计使用的情况。以鸟哥来说，我的数值模式仿真平台随便一个文件都好几百 MB，那么 block 容量当然选择较大的！至少文件系统就不必记录太多的 block 号码，读写起来也比较方便啊！



Tips 事实上，现在的磁盘容量都太大了！所以，大概大家都只会选择 4K 的 block 大小吧！呵呵！

- inode table （inode 表格）

再来讨论一下 inode 这个玩意儿吧！如前所述 inode 的内容在记录文件的属性以及该文件实际数据是放置在哪几号 block 内！ 基本上，inode 记录的文件数据至少有下面这些：[4]

- 该文件的存取模式（read/write/excute）；
- 该文件的拥有者与群组（owner/group）；
- 该文件的容量；
- 该文件创建或状态改变的时间（ctime）；
- 最近一次的读取时间（atime）；
- 最近修改的时间（mtime）；
- 定义文件特性的旗标（flag），如 SetUID...；
- 该文件真正内容的指向 （pointer）；

inode 的数量与大小也是在格式化时就已经固定了，除此之外 inode 还有些什么特色呢？

- 每个 inode 大小均固定为 128 Bytes （新的 ext4 与 xfs 可设置到 256 Bytes）；
- 每个文件都仅会占用一个 inode 而已；
- 承上，因此文件系统能够创建的文件数量与 inode 的数量有关；
- 系统读取文件时需要先找到 inode，并分析 inode 所记录的权限与使用者是否符合，若符合才能够开始实际读取 block 的内容。

我们约略来分析一下 EXT2 的 inode / block 与文件大小的关系好了。inode 要记录的数据非常多，但偏偏又只有 128Bytes 而已， 而 inode 记录一个 block 号码要花掉 4Byte ，假设我一个文件有 400MB 且每个 block 为 4K 时， 那么至少也要十万笔 block 号码的记录呢！inode 哪有这么多可记录的信息？为此我们的系统很聪明的将 inode 记录 block 号码的区域定义为12个直接，一个间接, 一个双间接与一个三间接记录区。这是啥？我们将 inode 的结构画一下好了。

图7.1.4、inode 结构示意图

上图最左边为 inode 本身 （128 Bytes），里面有 12 个直接指向 block 号码的对照，这 12 笔记录就能够直接取得 block 号码啦！ 至于所谓的间接就是再拿一个 block 来当作记录 block 号码的记录区，如果文件太大时， 就会使用间接的 block 来记录号码。如上图 7.1.4 当中间接只是拿一个 block 来记录额外的号码而已。 同理，如果文件持续长大，那么就会利用所谓的双间接，第一个 block 仅再指出下一个记录号码的 block 在哪里， 实际记录的在第二个 block 当中。依此类推，三间接就是利用第三层 block 来记录号码啦！

这样子 inode 能够指定多少个 block 呢？我们以较小的 1K block 来说明好了，可以指定的情况如下：

- 12 个直接指向： 12*1K=12K 由于是直接指向，所以总共可记录 12 笔记录，因此总额大小为如上所示；
- 间接： 256*1K=256K 每笔 block 号码的记录会花去 4Bytes，因此 1K 的大小能够记录 256 笔记录，因此一个间接可以记录的文件大小如上；
- 双间接： 256*256*1K=2562K 第一层 block 会指定 256 个第二层，每个第二层可以指定 256 个号码，因此总额大小如上；
- 三间接： 256*256*256*1K=2563K 第一层 block 会指定 256 个第二层，每个第二层可以指定 256 个第三层，每个第三层可以指定 256 个号码，因此总额大小如上；
- 总额：将直接、间接、双间接、三间接加总，得到 12 + 256 + 256*256 + 256*256*256 （K） = 16GB

此时我们知道当文件系统将 block 格式化为 1K 大小时，能够容纳的最大文件为 16GB，比较一下文件系统限制表的结果可发现是一致的！但这个方法不能用在 2K 及 4K block 大小的计算中， 因为大于 2K 的 block 将会受到 Ext2 文件系统本身的限制，所以计算的结果会不太符合之故。



Tips 如果你的 Linux 依旧使用 Ext2/Ext3/Ext4 文件系统的话，例如鸟哥之前的 CentOS 6.x 系统，那么默认还是使用 Ext4 的文件系统喔！ Ext4 文件系统的 inode 容量已经可以扩大到 256Bytes 了，更大的 inode 容量，可以纪录更多的文件系统信息，包括新的 ACL 以及 SELinux 类型等， 当然，可以纪录的单一文件大小达 16TB 且单一文件系统总容量可达 1EB 哩！

- Superblock （超级区块）

Superblock 是记录整个 filesystem 相关信息的地方， 没有 Superblock ，就没有这个 filesystem 了。他记录的信息主要有：

- block 与 inode 的总量；
- 未使用与已使用的 inode / block 数量；
- block 与 inode 的大小 （block 为 1, 2, 4K，inode 为 128Bytes 或 256Bytes）；
- filesystem 的挂载时间、最近一次写入数据的时间、最近一次检验磁盘 （fsck） 的时间等文件系统的相关信息；
- 一个 valid bit 数值，若此文件系统已被挂载，则 valid bit 为 0 ，若未被挂载，则 valid bit 为 1 。

Superblock 是非常重要的，因为我们这个文件系统的基本信息都写在这里，因此，如果 superblock 死掉了， 你的文件系统可能就需要花费很多时间去挽救啦！一般来说， superblock 的大小为 1024Bytes。相关的 superblock 讯息我们等一下会以 dumpe2fs 指令来调用出来观察喔！

此外，每个 block group 都可能含有 superblock 喔！但是我们也说一个文件系统应该仅有一个 superblock 而已，那是怎么回事啊？ 事实上除了第一个 block group 内会含有 superblock 之外，后续的 block group 不一定含有 superblock ， 而若含有 superblock 则该 superblock 主要是做为第一个 block group 内 superblock 的备份咯，这样可以进行 superblock 的救援呢！

- Filesystem Description （文件系统描述说明）

这个区段可以描述每个 block group 的开始与结束的 block 号码，以及说明每个区段 （superblock, bitmap, inodemap, data block） 分别介于哪一个 block 号码之间。这部份也能够用 dumpe2fs 来观察的。

- block bitmap （区块对照表）

如果你想要新增文件时总会用到 block 吧！那你要使用哪个 block 来记录呢？当然是选择“空的 block ”来记录新文件的数据啰。 那你怎么知道哪个 block 是空的？这就得要通过 block bitmap 的辅助了。从 block bitmap 当中可以知道哪些 block 是空的，因此我们的系统就能够很快速的找到可使用的空间来处置文件啰。

同样的，如果你删除某些文件时，那么那些文件原本占用的 block 号码就得要释放出来， 此时在 block bitmap 当中相对应到该 block 号码的标志就得要修改成为“未使用中”啰！这就是 bitmap 的功能。

- inode bitmap （inode 对照表）

这个其实与 block bitmap 是类似的功能，只是 block bitmap 记录的是使用与未使用的 block 号码， 至于 inode bitmap 则是记录使用与未使用的 inode 号码啰！

- dumpe2fs： 查询 Ext 家族 superblock 信息的指令

了解了文件系统的概念之后，再来当然是观察这个文件系统啰！刚刚谈到的各部分数据都与 block 号码有关！ 每个区段与 superblock 的信息都可以使用 dumpe2fs 这个指令来查询的！ 不过很可惜的是，我们的 CentOS 7 现在是以 xfs 为默认文件系统， 所以目前你的系统应该无法使用 dumpe2fs 去查询任何文件系统的。没关系，鸟哥先找自己的一部机器来跟大家介绍， 你可以在后续的格式化内容讲完之后，自己切出一个 ext4 的文件系统去查询看看即可。 鸟哥这块文件系统是 1GB 的容量，使用默认方式来进行格式化的， 观察的内容如下：

```
[root@study ~]# dumpe2fs [-bh] 设备文件名
选项与参数：
-b ：列出保留为坏轨的部分（一般用不到吧！？）
-h ：仅列出 superblock 的数据，不会列出其他的区段内容！

范例：鸟哥的一块 1GB ext4 文件系统内容
[root@study ~]# blkid   &lt;==这个指令可以叫出目前系统有被格式化的设备
/dev/vda1: LABEL="myboot" UUID="ce4dbf1b-2b3d-4973-8234-73768e8fd659" TYPE="xfs"
/dev/vda2: LABEL="myroot" UUID="21ad8b9a-aaad-443c-b732-4e2522e95e23" TYPE="xfs"
/dev/vda3: UUID="12y99K-bv2A-y7RY-jhEW-rIWf-PcH5-SaiApN" TYPE="LVM2_member"
/dev/vda5: UUID="e20d65d9-20d4-472f-9f91-cdcfb30219d6" TYPE="ext4"  &lt;==看到 ext4 了！

[root@study ~]# dumpe2fs /dev/vda5
dumpe2fs 1.42.9 （28-Dec-2013）
Filesystem volume name:   &lt;none&gt;           # 文件系统的名称（不一定会有）
Last mounted on:          &lt;not available&gt;  # 上一次挂载的目录位置
Filesystem UUID:          e20d65d9-20d4-472f-9f91-cdcfb30219d6
Filesystem magic number:  0xEF53            # 上方的 UUID 为 Linux 对设备的定义码
Filesystem revision #:    1 （dynamic）      # 下方的 features 为文件系统的特征数据
Filesystem features:      has_journal ext_attr resize_inode dir_index filetype extent 64b
 flex_bg sparse_super large_file huge_file uninit_bg dir_nlink extra_isize
Filesystem flags:         signed_directory_hash
Default mount options:    user_xattr acl    # 默认在挂载时会主动加上的挂载参数
Filesystem state:         clean             # 这块文件系统的状态为何，clean 是没问题
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              65536             # inode 的总数
Block count:              262144            # block 的总数
Reserved block count:     13107             # 保留的 block 总数
Free blocks:              249189            # 还有多少的 block 可用数量
Free inodes:              65525             # 还有多少的 inode 可用数量
First block:              0
Block size:               4096              # 单个 block 的容量大小
Fragment size:            4096
Group descriptor size:    64
....（中间省略）....
Inode size:               256               # inode 的容量大小！已经是 256 了喔！
....（中间省略）....
Journal inode:            8
Default directory hash:   half_md4
Directory Hash Seed:      3c2568b4-1a7e-44cf-95a2-c8867fb19fbc
Journal backup:           inode blocks
Journal features:         （none）
Journal size:             32M               # Journal 日志式数据的可供纪录总容量
Journal length:           8192
Journal sequence:         0x00000001
Journal start:            0

Group 0: （Blocks 0-32767）                   # 第一块 block group 位置
  Checksum 0x13be, unused inodes 8181
  Primary superblock at 0, Group descriptors at 1-1   # 主要 superblock 的所在喔！
  Reserved GDT blocks at 2-128
  Block bitmap at 129 （+129）, Inode bitmap at 145 （+145）
  Inode table at 161-672 （+161）                   # inode table 的所在喔！
  28521 free blocks, 8181 free inodes, 2 directories, 8181 unused inodes
  Free blocks: 142-144, 153-160, 4258-32767         # 下面两行说明剩余的容量有多少
  Free inodes: 12-8192
Group 1: （Blocks 32768-65535） [INODE_UNINIT]         # 后续为更多其他的 block group 喔！
....（下面省略）....
# 由于数据量非常的庞大，因此鸟哥将一些信息省略输出了！上表与你的屏幕会有点差异。
# 前半部在秀出 supberblock 的内容，包括标头名称（Label）以及inode/block的相关信息
# 后面则是每个 block group 的个别信息了！您可以看到各区段数据所在的号码！
# 也就是说，基本上所有的数据还是与 block 的号码有关就是了！很重要！
```

如上所示，利用 dumpe2fs 可以查询到非常多的信息，不过依内容主要可以区分为上半部是 superblock 内容， 下半部则是每个 block group 的信息了。从上面的表格中我们可以观察到鸟哥这个 /dev/vda5 规划的 block 为 4K， 第一个 block 号码为 0 号，且 block group 内的所有信息都以 block 的号码来表示的。 然后在 superblock 中还有谈到目前这个文件系统的可用 block 与 inode 数量喔！

至于 block group 的内容我们单纯看 Group0 信息好了。从上表中我们可以发现：

- Group0 所占用的 block 号码由 0 到 32767 号，superblock 则在第 0 号的 block 区块内！
- 文件系统描述说明在第 1 号 block 中；
- block bitmap 与 inode bitmap 则在 129 及 145 的 block 号码上。
- 至于 inode table 分布于 161-672 的 block 号码中！
- 由于 （1）一个 inode 占用 256 Bytes ，（2）总共有 672 - 161 + 1（161本身） = 512 个 block 花在 inode table 上， （3）每个 block 的大小为 4096 Bytes（4K） 。由这些数据可以算出 inode 的数量共有 512 * 4096 / 256 = 8192 个 inode 啦！
- 这个 Group0 目前可用的 block 有 28521 个，可用的 inode 有 8181 个；
- 剩余的 inode 号码为 12 号到 8192 号。

如果你对文件系统的详细信息还有更多想要了解的话，那么请参考本章最后一小节的介绍喔！ 否则文件系统看到这里对于基础认知您应该是已经相当足够啦！下面则是要探讨一下，那么这个文件系统概念与实际的目录树应用有啥关连啊？

7.1.4 与目录树的关系

由前一小节的介绍我们知道在 Linux 系统下，每个文件（不管是一般文件还是目录文件）都会占用一个 inode ， 且可依据文件内容的大小来分配多个 block 给该文件使用。而由第五章的权限说明中我们知道目录的内容在记录文件名， 一般文件才是实际记录数据内容的地方。那么目录与文件在文件系统当中是如何记录数据的呢？基本上可以这样说：

- 目录

当我们在 Linux 下的文件系统创建一个目录时，文件系统会分配一个 inode 与至少一块 block 给该目录。其中，inode 记录该目录的相关权限与属性，并可记录分配到的那块 block 号码； 而 block 则是记录在这个目录下的文件名与该文件名占用的 inode 号码数据。也就是说目录所占用的 block 内容在记录如下的信息：

Inode number	檔名
53735697 53745858 ...	anaconda-ks.cfg initial-setup-ks.cfg ...

图7.1.5、记载于目录所属的 block 内的文件名与 inode 号码对应示意图

如果想要实际观察 root 主文件夹内的文件所占用的 inode 号码时，可以使用 ls -i 这个选项来处理：

```
[root@study ~]# ls -li
total 8
53735697 -rw-------. 1 root root 1816 May  4 17:57 anaconda-ks.cfg
53745858 -rw-r--r--. 1 root root 1864 May  4 18:01 initial-setup-ks.cfg
```

由于每个人所使用的计算机并不相同，系统安装时选择的项目与 partition 都不一样，因此你的环境不可能与我的 inode 号码一模一样！上表的左边所列出的 inode 仅是鸟哥的系统所显示的结果而已！而由这个目录的 block 结果我们现在就能够知道， 当你使用“ ll / ”时，出现的目录几乎都是 1024 的倍数，为什么呢？因为每个 block 的数量都是 1K, 2K, 4K 嘛！ 看一下鸟哥的环境：

```
[root@study ~]# ll -d / /boot /usr/sbin /proc /sys
dr-xr-xr-x.  17 root root  4096 May  4 17:56 /         &lt;== 1 个 4K block
dr-xr-xr-x.   4 root root  4096 May  4 17:59 /boot     &lt;== 1 个 4K block
dr-xr-xr-x. 155 root root     0 Jun 15 15:43 /proc     &lt;== 这两个为内存内数据，不占磁盘容量
dr-xr-xr-x.  13 root root     0 Jun 15 23:43 /sys
dr-xr-xr-x.   2 root root 16384 May  4 17:55 /usr/sbin &lt;== 4 个 4K block
```

由于鸟哥的根目录使用的 block 大小为 4K ，因此每个目录几乎都是 4K 的倍数。 其中由于 /usr/sbin 的内容比较复杂因此占用了 4 个 block ！至于奇怪的 /proc 我们在第五章就讲过该目录不占磁盘容量， 所以当然耗用的 block 就是 0 啰！



Tips 由上面的结果我们知道目录并不只会占用一个 block 而已，也就是说： 在目录下面的文件数如果太多而导致一个 block 无法容纳的下所有的文件名与 inode 对照表时，Linux 会给予该目录多一个 block 来继续记录相关的数据；

- 文件：

当我们在 Linux 下的 ext2 创建一个一般文件时， ext2 会分配一个 inode 与相对于该文件大小的 block 数量给该文件。例如：假设我的一个 block 为 4 KBytes ，而我要创建一个 100 KBytes 的文件，那么 linux 将分配一个 inode 与 25 个 block 来储存该文件！ 但同时请注意，由于 inode 仅有 12 个直接指向，因此还要多一个 block 来作为区块号码的记录喔！

- 目录树读取：

好了，经过上面的说明你也应该要很清楚的知道 inode 本身并不记录文件名，文件名的记录是在目录的 block 当中。 因此在第五章文件与目录的权限说明中， 我们才会提到“新增/删除/更名文件名与目录的 w 权限有关”的特色！那么因为文件名是记录在目录的 block 当中， 因此

当我们要读取某个文件时，就务必会经过目录的 inode 与 block ，然后才能够找到那个待读取文件的 inode 号码， 最终才会读到正确的文件的 block 内的数据。

由于目录树是由根目录开始读起，因此系统通过挂载的信息可以找到挂载点的 inode 号码，此时就能够得到根目录的 inode 内容，并依据该 inode 读取根目录的 block 内的文件名数据，再一层一层的往下读到正确的文件名。举例来说，如果我想要读取 /etc/passwd 这个文件时，系统是如何读取的呢？

```
[root@study ~]# ll -di / /etc /etc/passwd
 128 dr-xr-xr-x.  17 root root 4096 May  4 17:56 /
33595521 drwxr-xr-x. 131 root root 8192 Jun 17 00:20 /etc
36628004 -rw-r--r--.   1 root root 2092 Jun 17 00:20 /etc/passwd
```

在鸟哥的系统上面与 /etc/passwd 有关的目录与文件数据如上表所示，该文件的读取流程为（假设读取者身份为 dmtsai 这个一般身份使用者）：

1. / 的 inode： 通过挂载点的信息找到 inode 号码为 128 的根目录 inode，且 inode 规范的权限让我们可以读取该 block 的内容（有 r 与 x）；
2. / 的 block： 经过上个步骤取得 block 的号码，并找到该内容有 etc/ 目录的 inode 号码（33595521）；
3. etc/ 的 inode： 读取 33595521 号 inode 得知 dmtsai 具有 r 与 x 的权限，因此可以读取 etc/ 的 block 内容；
4. etc/ 的 block： 经过上个步骤取得 block 号码，并找到该内容有 passwd 文件的 inode 号码 （36628004）；
5. passwd 的 inode： 读取 36628004 号 inode 得知 dmtsai 具有 r 的权限，因此可以读取 passwd 的 block 内容；
6. passwd 的 block： 最后将该 block 内容的数据读出来。
7. filesystem 大小与磁盘读取性能：

另外，关于文件系统的使用效率上，当你的一个文件系统规划的很大时，例如 100GB 这么大时， 由于磁盘上面的数据总是来来去去的，所以，整个文件系统上面的文件通常无法连续写在一起（block 号码不会连续的意思）， 而是填入式的将数据填入没有被使用的 block 当中。如果文件写入的 block 真的分的很散， 此时就会有所谓的文件数据离散的问题发生了。

如前所述，虽然我们的 ext2 在 inode 处已经将该文件所记录的 block 号码都记上了， 所以数据可以一次性读取，但是如果文件真的太过离散，确实还是会发生读取效率低落的问题。 因为磁头还是得要在整个文件系统中来来去去的频繁读取！ 果真如此，那么可以将整个 filesystme 内的数据全部复制出来，将该 filesystem 重新格式化， 再将数据给他复制回去即可解决这个问题。

此外，如果 filesystem 真的太大了，那么当一个文件分别记录在这个文件系统的最前面与最后面的 block 号码中， 此时会造成磁盘的机械手臂移动幅度过大，也会造成数据读取性能的低落。而且磁头在搜寻整个 filesystem 时， 也会花费比较多的时间去搜寻！因此， partition 的规划并不是越大越好， 而是真的要针对您的主机用途来进行规划才行！^_^

7.1.5 EXT2/EXT3/EXT4 文件的存取与日志式文件系统的功能

上一小节谈到的仅是读取而已，那么如果是新建一个文件或目录时，我们的文件系统是如何处理的呢？ 这个时候就得要 block bitmap 及 inode bitmap 的帮忙了！假设我们想要新增一个文件，此时文件系统的行为是：

1. 先确定使用者对于欲新增文件的目录是否具有 w 与 x 的权限，若有的话才能新增；
2. 根据 inode bitmap 找到没有使用的 inode 号码，并将新文件的权限/属性写入；
3. 根据 block bitmap 找到没有使用中的 block 号码，并将实际的数据写入 block 中，且更新 inode 的 block 指向数据；
4. 将刚刚写入的 inode 与 block 数据同步更新 inode bitmap 与 block bitmap，并更新 superblock 的内容。

一般来说，我们将 inode table 与 data block 称为数据存放区域，至于其他例如 superblock、 block bitmap 与 inode bitmap 等区段就被称为 metadata （中介数据） 啰，因为 superblock, inode bitmap 及 block bitmap 的数据是经常变动的，每次新增、移除、编辑时都可能会影响到这三个部分的数据，因此才被称为中介数据的啦。

- 数据的不一致 （Inconsistent） 状态

在一般正常的情况下，上述的新增动作当然可以顺利的完成。但是如果有个万一怎么办？ 例如你的文件在写入文件系统时，因为不知名原因导致系统中断（例如突然的停电啊、 系统核心发生错误啊～等等的怪事发生时），所以写入的数据仅有 inode table 及 data block 而已，最后一个同步更新中介数据的步骤并没有做完，此时就会发生 metadata 的内容与实际数据存放区产生不一致 （Inconsistent） 的情况了。

既然有不一致当然就得要克服！在早期的 Ext2 文件系统中，如果发生这个问题， 那么系统在重新开机的时候，就会借由 Superblock 当中记录的 valid bit （是否有挂载） 与 filesystem state （clean 与否） 等状态来判断是否强制进行数据一致性的检查！若有需要检查时则以 e2fsck 这支程序来进行的。

不过，这样的检查真的是很费时～因为要针对 metadata 区域与实际数据存放区来进行比对， 呵呵～得要搜寻整个 filesystem 呢～如果你的文件系统有 100GB 以上，而且里面的文件数量又多时， 哇！系统真忙碌～而且在对 Internet 提供服务的服务器主机上面， 这样的检查真的会造成主机复原时间的拉长～真是麻烦～这也就造成后来所谓日志式文件系统的兴起了。

- 日志式文件系统 （Journaling filesystem）

为了避免上述提到的文件系统不一致的情况发生，因此我们的前辈们想到一个方式， 如果在我们的 filesystem 当中规划出一个区块，该区块专门在记录写入或修订文件时的步骤， 那不就可以简化一致性检查的步骤了？也就是说：

1. 预备：当系统要写入一个文件时，会先在日志记录区块中纪录某个文件准备要写入的信息；
2. 实际写入：开始写入文件的权限与数据；开始更新 metadata 的数据；
3. 结束：完成数据与 metadata 的更新后，在日志记录区块当中完成该文件的纪录。

在这样的程序当中，万一数据的纪录过程当中发生了问题，那么我们的系统只要去检查日志记录区块， 就可以知道哪个文件发生了问题，针对该问题来做一致性的检查即可，而不必针对整块 filesystem 去检查， 这样就可以达到快速修复 filesystem 的能力了！这就是日志式文件最基础的功能啰～

那么我们的 ext2 可达到这样的功能吗？当然可以啊！ 就通过 ext3/ext4 即可！ ext3/ext4 是 ext2 的升级版本，并且可向下相容 ext2 版本呢！ 所以啰，目前我们才建议大家，可以直接使用 ext4 这个 filesystem 啊！ 如果你还记得 dumpe2fs 输出的讯息，可以发现 superblock 里面含有下面这样的信息：

```
Journal inode:            8
Journal backup:           inode blocks
Journal features:         （none）
Journal size:             32M
Journal length:           8192
Journal sequence:         0x00000001
```

看到了吧！通过 inode 8 号记录 journal 区块的 block 指向，而且具有 32MB 的容量在处理日志呢！ 这样对于所谓的日志式文件系统有没有比较有概念一点呢？^_^。

7.1.6 Linux 文件系统的运行

我们现在知道了目录树与文件系统的关系了，但是由第零章的内容我们也知道， 所有的数据都得要载入到内存后 CPU 才能够对该数据进行处理。想一想，如果你常常编辑一个好大的文件， 在编辑的过程中又频繁的要系统来写入到磁盘中，由于磁盘写入的速度要比内存慢很多， 因此你会常常耗在等待磁盘的写入/读取上。真没效率！

为了解决这个效率的问题，因此我们的 Linux 使用的方式是通过一个称为非同步处理（asynchronously） 的方式。所谓的非同步处理是这样的：

当系统载入一个文件到内存后，如果该文件没有被更动过，则在内存区段的文件数据会被设置为干净（clean）的。 但如果内存中的文件数据被更改过了（例如你用 nano 去编辑过这个文件），此时该内存中的数据会被设置为脏的 （Dirty）。此时所有的动作都还在内存中执行，并没有写入到磁盘中！ 系统会不定时的将内存中设置为“Dirty”的数据写回磁盘，以保持磁盘与内存数据的一致性。 你也可以利用第四章谈到的 sync指令来手动强迫写入磁盘。

我们知道内存的速度要比磁盘快的多，因此如果能够将常用的文件放置到内存当中，这不就会增加系统性能吗？ 没错！是有这样的想法！因此我们 Linux 系统上面文件系统与内存有非常大的关系喔：

- 系统会将常用的文件数据放置到内存的缓冲区，以加速文件系统的读/写；
- 承上，因此 Linux 的实体内存最后都会被用光！这是正常的情况！可加速系统性能；
- 你可以手动使用 sync 来强迫内存中设置为 Dirty 的文件回写到磁盘中；
- 若正常关机时，关机指令会主动调用 sync 来将内存的数据回写入磁盘内；
- 但若不正常关机（如跳电、死机或其他不明原因），由于数据尚未回写到磁盘内， 因此重新开机后可能会花很多时间在进行磁盘检验，甚至可能导致文件系统的损毁（非磁盘损毁）。

7.1.7 挂载点的意义 （mount point）

每个 filesystem 都有独立的 inode / block / superblock 等信息，这个文件系统要能够链接到目录树才能被我们使用。 将文件系统与目录树结合的动作我们称为“挂载”。 关于挂载的一些特性我们在第二章稍微提过， 重点是：挂载点一定是目录，该目录为进入该文件系统的入口。 因此并不是你有任何文件系统都能使用，必须要“挂载”到目录树的某个目录后，才能够使用该文件系统的。

举例来说，如果你是依据鸟哥的方法安装你的 CentOS 7.x 的话， 那么应该会有三个挂载点才是，分别是 /, /boot, /home 三个 （鸟哥的系统上对应的设备文件名为 LVM, LVM, /dev/vda2）。 那如果观察这三个目录的 inode 号码时，我们可以发现如下的情况：

```
[root@study ~]# ls -lid / /boot /home
128 dr-xr-xr-x. 17 root root 4096 May  4 17:56 /
128 dr-xr-xr-x.  4 root root 4096 May  4 17:59 /boot
128 drwxr-xr-x.  5 root root   41 Jun 17 00:20 /home
```

看到了吧！由于 XFS filesystem 最顶层的目录之 inode 一般为 128 号，因此可以发现 /, /boot, /home 为三个不同的 filesystem 啰！ （因为每一行的文件属性并不相同，且三个目录的挂载点也均不相同之故。） 我们在第六章一开始的路径中曾经提到根目录下的 . 与 .. 是相同的东西， 因为权限是一模一样嘛！如果使用文件系统的观点来看，同一个 filesystem 的某个 inode 只会对应到一个文件内容而已（因为一个文件占用一个 inode 之故）， 因此我们可以通过判断 inode 号码来确认不同文件名是否为相同的文件喔！所以可以这样看：

```
[root@study ~]# ls -ild /  /.  /..
128 dr-xr-xr-x. 17 root root 4096 May  4 17:56 /
128 dr-xr-xr-x. 17 root root 4096 May  4 17:56 /.
128 dr-xr-xr-x. 17 root root 4096 May  4 17:56 /..
```

上面的信息中由于挂载点均为 / ，因此三个文件 （/, /., /..） 均在同一个 filesystem 内，而这三个文件的 inode 号码均为 128 号，因此这三个文件名都指向同一个 inode 号码，当然这三个文件的内容也就完全一模一样了！ 也就是说，根目录的上层 （/..） 就是他自己！这么说，

看的懂了吗？ ^_^

7.1.8 其他 Linux 支持的文件系统与 VFS

虽然 Linux 的标准文件系统是 ext2 ，且还有增加了日志功能的 ext3/ext4 ，事实上，Linux 还有支持很多文件系统格式的， 尤其是最近这几年推出了好几种速度很快的日志式文件系统， 包括 SGI 的 XFS 文件系统， 可以适用更小型文件的 Reiserfs 文件系统，以及 Windows 的 FAT 文件系统等等， 都能够被 Linux 所支持喔！常见的支持文件系统有：

- 传统文件系统：ext2 / minix / MS-DOS / FAT （用 vfat 模块） / iso9660 （光盘）等等；
- 日志式文件系统： ext3 /ext4 / ReiserFS / Windows' NTFS / IBM's JFS / SGI's XFS / ZFS
- 网络文件系统： NFS / SMBFS

想要知道你的 Linux 支持的文件系统有哪些，可以察看下面这个目录：

```
[root@study ~]# ls -l /lib/modules/$（uname -r）/kernel/fs
```

系统目前已载入到内存中支持的文件系统则有：

```
[root@study ~]# cat /proc/filesystems
```

- Linux VFS （Virtual Filesystem Switch）

了解了我们使用的文件系统之后，再来则是要提到，那么 Linux 的核心又是如何管理这些认识的文件系统呢？ 其实，整个 Linux 的系统都是通过一个名为 Virtual Filesystem Switch 的核心功能去读取 filesystem 的。 也就是说，整个 Linux 认识的 filesystem 其实都是 VFS 在进行管理，我们使用者并不需要知道每个 partition 上头的 filesystem 是什么～ VFS 会主动的帮我们做好读取的动作呢～

假设你的 / 使用的是 /dev/hda1 ，用 ext3 ，而 /home 使用 /dev/hda2 ，用 reiserfs ， 那么你取用 /home/dmtsai/.bashrc 时，有特别指定要用的什么文件系统的模块来读取吗？ 应该是没有吧！这个就是 VFS 的功能啦！通过这个 VFS 的功能来管理所有的 filesystem， 省去我们需要自行设置读取文件系统的定义啊～方便很多！整个 VFS 可以约略用下图来说明：

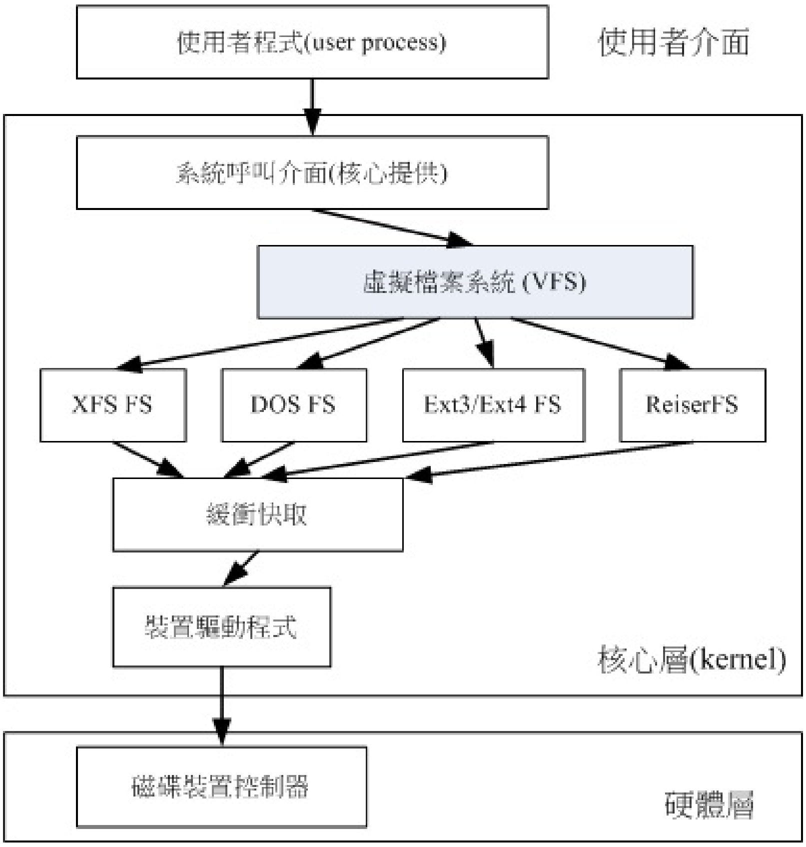


图7.1.6、VFS 文件系统的示意图

老实说，文件系统真的不好懂！ 如果你想要对文件系统有更深入的了解，文末的相关链接[5]务必要参考参考才好喔！

7.1.9 XFS 文件系统简介

CentOS 7 开始，默认的文件系统已经由原本的 EXT4 变成了 XFS 文件系统了！为啥 CentOS 要舍弃对 Linux 支持度最完整的 EXT 家族而改用 XFS 呢？ 这是有一些原因存在的。

- EXT 家族当前较伤脑筋的地方：支持度最广，但格式化超慢！

Ext 文件系统家族对于文件格式化的处理方面，采用的是预先规划出所有的 inode/block/meta data 等数据，未来系统可以直接取用， 不需要再进行动态配置的作法。这个作法在早期磁盘容量还不大的时候还算 OK 没啥问题，但时至今日，磁盘容量越来越大，连传统的 MBR 都已经被 GPT 所取代，连我们这些老人家以前听到的超大 TB 容量也已经不够看了！现在都已经说到 PB 或 EB 以上容量了呢！那你可以想像得到，当你的 TB 以上等级的传统 ext 家族文件系统在格式化的时候，光是系统要预先分配 inode 与 block 就消耗你好多好多的人类时间了...



Tips 之前格式化过一个 70 TB 以上的磁盘阵列成为 ext4 文件系统，按下格式化，去喝了咖啡、吃了便当才回来看做完了没有... 所以，后来立刻改成 xfs 文件系统了。

另外，由于虚拟化的应用越来越广泛，而作为虚拟化磁盘来源的巨型文件 （单一文件好几个 GB 以上！） 也就越来越常见了。 这种巨型文件在处理上需要考虑到性能问题，否则虚拟磁盘的效率就会不太好看。因此，从 CentOS 7.x 开始， 文件系统已经由默认的 Ext4 变成了 xfs 这一个较适合大容量磁盘与巨型文件性能较佳的文件系统了。



Tips 其实鸟哥有几组虚拟计算机教室服务器系统，里面跑的确实是 EXT4 文件系统，老实说，并不觉得比 xfs 慢！所以，对鸟哥来说， 性能并不是主要改变文件系统的考虑！对于文件系统的复原速度、创建速度，可能才是鸟哥改换成 xfs 的思考点。

- XFS 文件系统的配置 [6]

基本上 xfs 就是一个日志式文件系统，而 CentOS 7.x 拿它当默认的文件系统，自然就是因为最早之前，这个 xfs 就是被开发来用于大容量磁盘以及高性能文件系统之用， 因此，相当适合现在的系统环境。此外，几乎所有 Ext4 文件系统有的功能， xfs 都可以具备！也因此在本小节前几部份谈到文件系统时， 其实大部份的操作依旧是在 xfs 文件系统环境下介绍给各位的哩！

xfs 文件系统在数据的分佈上，主要规划为三个部份，一个数据区 （data section） 、一个文件系统活动登录区 （log section）以及一个实时运行区 （realtime section）。 这三个区域的数据内容如下：

- 数据区 （data section）

基本上，数据区就跟我们之前谈到的 ext 家族一样，包括 inode/data block/superblock 等数据，都放置在这个区块。 这个数据区与 ext 家族的 block group 类似，也是分为多个储存区群组 （allocation groups） 来分别放置文件系统所需要的数据。 每个储存区群组都包含了 （1）整个文件系统的 superblock、 （2）剩余空间的管理机制、 （3）inode的分配与追踪。此外，inode与 block 都是系统需要用到时， 这才动态配置产生，所以格式化动作超级快！

另外，与 ext 家族不同的是， xfs 的 block 与 inode 有多种不同的容量可供设置，block 容量可由 512Bytes ~ 64K 调配，不过，Linux 的环境下， 由于内存控制的关系 （分页档 pagesize 的容量之故），因此最高可以使用的 block 大小为 4K 而已！（鸟哥尝试格式化 block 成为 16K 是没问题的，不过，Linux 核心不给挂载！ 所以格式化完成后也无法使用啦！） 至于 inode 容量可由 256Bytes 到 2M 这么大！不过，大概还是保留 256Bytes 的默认值就很够用了！



Tips 总之， xfs 的这个数据区的储存区群组 （allocation groups, AG），你就将它想成是 ext 家族的 block 群组 （block groups） 就对了！本小节之前讲的都可以在这个区块内使用。 只是 inode 与 block 是动态产生，并非一开始于格式化就完成配置的。

- 文件系统活动登录区 （log section）

在登录区这个区域主要被用来纪录文件系统的变化，其实有点像是日志区啦！文件的变化会在这里纪录下来，直到该变化完整的写入到数据区后， 该笔纪录才会被终结。如果文件系统因为某些缘故 （例如最常见的停电） 而损毁时，系统会拿这个登录区块来进行检验，看看系统挂掉之前， 文件系统正在运行些啥动作，借以快速的修复文件系统。

因为系统所有动作的时候都会在这个区块做个纪录，因此这个区块的磁盘活动是相当频繁的！xfs 设计有点有趣，在这个区域中， 你可以指定外部的磁盘来作为 xfs 文件系统的日志区块喔！例如，你可以将 SSD 磁盘作为 xfs 的登录区，这样当系统需要进行任何活动时， 就可以更快速的进行工作！相当有趣！

- 实时运行区 （realtime section）

当有文件要被创建时，xfs 会在这个区段里面找一个到数个的 extent 区块，将文件放置在这个区块内，等到分配完毕后，再写入到 data section 的 inode 与 block 去！ 这个 extent 区块的大小得要在格式化的时候就先指定，最小值是 4K 最大可到 1G。一般非磁盘阵列的磁盘默认为 64K 容量，而具有类似磁盘阵列的 stripe 情况下，则建议 extent 设置为与 stripe 一样大较佳。这个 extent 最好不要乱动，因为可能会影响到实体磁盘的性能喔。

- XFS 文件系统的描述数据观察

刚刚讲了这么多，完全无法理会耶～有没有像 EXT 家族的 dumpe2fs 去观察 superblock 内容的相关指令可以查阅呢？有啦！可以使用 xfs_info 去观察的！ 详细的指令作法可以参考如下：

```
[root@study ~]# xfs_info 挂载点&#124;设备文件名

范例一：找出系统 /boot 这个挂载点下面的文件系统的 superblock 纪录
[root@study ~]# df -T /boot
Filesystem     Type 1K-blocks   Used Available Use% Mounted on
/dev/vda2      xfs    1038336 133704    904632  13% /boot
# 没错！可以看得出来是 xfs 文件系统的！来观察一下内容吧！

[root@study ~]# xfs_info /dev/vda2
1  meta-data=/dev/vda2         isize=256    agcount=4, agsize=65536 blks
2           =                  sectsz=512   attr=2, projid32bit=1
3           =                  crc=0        finobt=0
4  data     =                  bsize=4096   blocks=262144, imaxpct=25
5           =                  sunit=0      swidth=0 blks
6  naming   =version 2         bsize=4096   ascii-ci=0 ftype=0
7  log      =internal          bsize=4096   blocks=2560, version=2
8           =                  sectsz=512   sunit=0 blks, lazy-count=1
9  realtime =none              extsz=4096   blocks=0, rtextents=0
```

上面的输出讯息可以这样解释：

- 第 1 行里面的 isize 指的是 inode 的容量，每个有 256Bytes 这么大。至于 agcount 则是前面谈到的储存区群组 （allocation group） 的个数，共有 4 个， agsize 则是指每个储存区群组具有 65536 个 block 。配合第 4 行的 block 设置为 4K，因此整个文件系统的容量应该就是 4*65536*4K 这么大！
- 第 2 行里面 sectsz 指的是逻辑扇区 （sector） 的容量设置为 512Bytes 这么大的意思。
- 第 4 行里面的 bsize 指的是 block 的容量，每个 block 为 4K 的意思，共有 262144 个 block 在这个文件系统内。
- 第 5 行里面的 sunit 与 swidth 与磁盘阵列的 stripe 相关性较高。这部份我们下面格式化的时候会举一个例子来说明。
- 第 7 行里面的 internal 指的是这个登录区的位置在文件系统内，而不是外部设备的意思。且占用了 4K * 2560 个 block，总共约 10M 的容量。
- 第 9 行里面的 realtime 区域，里面的 extent 容量为 4K。不过目前没有使用。

由于我们并没有使用磁盘阵列，因此上头这个设备里头的 sunit 与 extent 就没有额外的指定特别的值。根据 xfs（5） 的说明，这两个值会影响到你的文件系统性能， 所以格式化的时候要特别留意喔！上面的说明大致上看看即可，比较重要的部份已经用特殊字体圈起来，你可以瞧一瞧先！

7.2 文件系统的简单操作

稍微了解了文件系统后，再来我们得要知道如何查询整体文件系统的总容量与每个目录所占用的容量啰！ 此外，前两章谈到的文件类型中尚未讲的很清楚的链接文件 （Link file） 也会在这一小节当中介绍的。

7.2.1 磁盘与目录的容量

现在我们知道磁盘的整体数据是在 superblock 区块中，但是每个各别文件的容量则在 inode 当中记载的。 那在命令行下面该如何叫出这几个数据呢？下面就让我们来谈一谈这两个指令：

- df：列出文件系统的整体磁盘使用量；
- du：评估文件系统的磁盘使用量（常用在推估目录所占容量）

- df

```
[root@study ~]# df [-ahikHTm] [目录或文件名]
选项与参数：
-a  ：列出所有的文件系统，包括系统特有的 /proc 等文件系统；
-k  ：以 KBytes 的容量显示各文件系统；
-m  ：以 MBytes 的容量显示各文件系统；
-h  ：以人们较易阅读的 GBytes, MBytes, KBytes 等格式自行显示；
-H  ：以 M=1000K 取代 M=1024K 的进位方式；
-T  ：连同该 partition 的 filesystem 名称 （例如 xfs） 也列出；
-i  ：不用磁盘容量，而以 inode 的数量来显示

范例一：将系统内所有的 filesystem 列出来！
[root@study ~]# df
Filesystem              1K-blocks    Used Available Use% Mounted on
/dev/mapper/centos-root  10475520 3409408   7066112  33% /
devtmpfs                   627700       0    627700   0% /dev
tmpfs                      637568      80    637488   1% /dev/shm
tmpfs                      637568   24684    612884   4% /run
tmpfs                      637568       0    637568   0% /sys/fs/cgroup
/dev/mapper/centos-home   5232640   67720   5164920   2% /home
/dev/vda2                 1038336  133704    904632  13% /boot
# 在 Linux 下面如果 df 没有加任何选项，那么默认会将系统内所有的
# （不含特殊内存内的文件系统与 swap） 都以 1 KBytes 的容量来列出来！
# 至于那个 /dev/shm 是与内存有关的挂载，先不要理他！
```

先来说明一下范例一所输出的结果讯息为：

- Filesystem：代表该文件系统是在哪个 partition ，所以列出设备名称；
- 1k-blocks：说明下面的数字单位是 1KB 呦！可利用 -h 或 -m 来改变容量；
- Used：顾名思义，就是使用掉的磁盘空间啦！
- Available：也就是剩下的磁盘空间大小；
- Use%：就是磁盘的使用率啦！如果使用率高达 90% 以上时， 最好需要注意一下了，免得容量不足造成系统问题喔！（例如最容易被灌爆的 /var/spool/mail 这个放置邮件的磁盘）

- Mounted on：就是磁盘挂载的目录所在啦！（挂载点啦！）

```
范例二：将容量结果以易读的容量格式显示出来
[root@study ~]# df -h
Filesystem               Size  Used Avail Use% Mounted on
/dev/mapper/centos-root   10G  3.3G  6.8G  33% /
devtmpfs                 613M     0  613M   0% /dev
tmpfs                    623M   80K  623M   1% /dev/shm
tmpfs                    623M   25M  599M   4% /run
tmpfs                    623M     0  623M   0% /sys/fs/cgroup
/dev/mapper/centos-home  5.0G   67M  5.0G   2% /home
/dev/vda2               1014M  131M  884M  13% /boot
# 不同于范例一，这里会以 G/M 等容量格式显示出来，比较容易看啦！

范例三：将系统内的所有特殊文件格式及名称都列出来
[root@study ~]# df -aT
Filesystem              Type        1K-blocks    Used Available Use% Mounted on
rootfs                  rootfs       10475520 3409368   7066152  33% /
proc                    proc                0       0         0    - /proc
sysfs                   sysfs               0       0         0    - /sys
devtmpfs                devtmpfs       627700       0    627700   0% /dev
securityfs              securityfs          0       0         0    - /sys/kernel/security
tmpfs                   tmpfs          637568      80    637488   1% /dev/shm
devpts                  devpts              0       0         0    - /dev/pts
tmpfs                   tmpfs          637568   24684    612884   4% /run
tmpfs                   tmpfs          637568       0    637568   0% /sys/fs/cgroup
.....（中间省略）.....
/dev/mapper/centos-root xfs          10475520 3409368   7066152  33% /
selinuxfs               selinuxfs           0       0         0    - /sys/fs/selinux
.....（中间省略）.....
/dev/mapper/centos-home xfs           5232640   67720   5164920   2% /home
/dev/vda2               xfs           1038336  133704    904632  13% /boot
binfmt_misc             binfmt_misc         0       0         0    - /proc/sys/fs/binfmt_
# 系统里面其实还有很多特殊的文件系统存在的。那些比较特殊的文件系统几乎
# 都是在内存当中，例如 /proc 这个挂载点。因此，这些特殊的文件系统
# 都不会占据磁盘空间喔！ ^_^

范例四：将 /etc 下面的可用的磁盘容量以易读的容量格式显示
[root@study ~]# df -h /etc
Filesystem               Size  Used Avail Use% Mounted on
/dev/mapper/centos-root   10G  3.3G  6.8G  33% /
# 这个范例比较有趣一点啦，在 df 后面加上目录或者是文件时， df
# 会自动的分析该目录或文件所在的 partition ，并将该 partition 的容量显示出来，
# 所以，您就可以知道某个目录下面还有多少容量可以使用了！ ^_^

范例五：将目前各个 partition 当中可用的 inode 数量列出
[root@study ~]# df -ih
Filesystem              Inodes IUsed IFree IUse% Mounted on
/dev/mapper/centos-root    10M  108K  9.9M    2% /
devtmpfs                  154K   397  153K    1% /dev
tmpfs                     156K     5  156K    1% /dev/shm
tmpfs                     156K   497  156K    1% /run
tmpfs                     156K    13  156K    1% /sys/fs/cgroup
# 这个范例则主要列出可用的 inode 剩余量与总容量。分析一下与范例一的关系，
# 你可以清楚的发现到，通常 inode 的数量剩余都比 block 还要多呢
```

由于 df 主要读取的数据几乎都是针对一整个文件系统，因此读取的范围主要是在 Superblock 内的信息， 所以这个指令显示结果的速度非常的快速！在显示的结果中你需要特别留意的是那个根目录的剩余容量！ 因为我们所有的数据都是由根目录衍生出来的，因此当根目录的剩余容量剩下 0 时，那你的 Linux 可能就问题很大了。



Tips 说个陈年老笑话！鸟哥还在念书时，别的研究室有个管理 Sun 工作站的研究生发现， 他的磁盘明明还有好几 GB ，但是就是没有办法将光盘内几 MB 的数据 copy 进去， 他就去跟老板讲说机器坏了！嘿！明明才来维护过几天而已为何会坏了！ 结果他老板就将维护商叫来骂了 2 小时左右吧！

后来，维护商发现原来磁盘的“总空间”还有很多， 只是某个分区填满了，偏偏该研究生就是要将数据 copy 去那个分区！呵呵！ 后来那个研究生就被命令“再也不许碰 Sun 主机”了～～

另外需要注意的是，如果使用 -a 这个参数时，系统会出现 /proc 这个挂载点，但是里面的东西都是 0 ，不要紧张！ /proc 的东西都是 Linux 系统所需要载入的系统数据，而且是挂载在“内存当中”的， 所以当然没有占任何的磁盘空间啰！

至于那个 /dev/shm/ 目录，其实是利用内存虚拟出来的磁盘空间，通常是总实体内存的一半！由于是通过内存仿真出来的磁盘，因此你在这个目录下面创建任何数据文件时，存取速度是非常快速的！（在内存内工作） 不过，也由于他是内存仿真出来的，因此这个文件系统的大小在每部主机上都不一样，而且创建的东西在下次开机时就消失了！ 因为是在内存中嘛！

- du

```
[root@study ~]# du [-ahskm] 文件或目录名称
选项与参数：
-a   ：列出所有的文件与目录容量，因为默认仅统计目录下面的文件量而已。
-h   ：以人们较易读的容量格式 （G/M） 显示；
-s   ：列出总量而已，而不列出每个各别的目录占用容量；
-S   ：不包括子目录下的总计，与 -s 有点差别。
-k   ：以 KBytes 列出容量显示；
-m   ：以 MBytes 列出容量显示；

范例一：列出目前目录下的所有文件大小
[root@study ~]# du
4       ./.cache/dconf  &lt;==每个目录都会列出来
4       ./.cache/abrt
8       ./.cache
....（中间省略）....
0       ./test4
4       ./.ssh          &lt;==包括隐藏文件的目录
76      .               &lt;==这个目录（.）所占用的总量
# 直接输入 du 没有加任何选项时，则 du 会分析“目前所在目录”
# 的文件与目录所占用的磁盘空间。但是，实际显示时，仅会显示目录容量（不含文件），
# 因此 . 目录有很多文件没有被列出来，所以全部的目录相加不会等于 . 的容量喔！
# 此外，输出的数值数据为 1K 大小的容量单位。

范例二：同范例一，但是将文件的容量也列出来
[root@study ~]# du -a
4       ./.bash_logout         &lt;==有文件的列表了
4       ./.bash_profile
4       ./.bashrc
....（中间省略）....
4       ./.ssh/known_hosts
4       ./.ssh
76      .

范例三：检查根目录下面每个目录所占用的容量
[root@study ~]# du -sm /*
0       /bin
99      /boot
....（中间省略）....
du: cannot access ‘/proc/17772/task/17772/fd/4’: No such file or directory
du: cannot access ‘/proc/17772/fdinfo/4’: No such file or directory
0       /proc      &lt;==不会占用硬盘空间！
1       /root
25      /run
....（中间省略）....
3126    /usr       &lt;==系统初期最大就是他了啦！
117     /var
# 这是个很常被使用的功能～利用万用字符 * 来代表每个目录，如果想要检查某个目录下，
# 哪个次目录占用最大的容量，可以用这个方法找出来。值得注意的是，如果刚刚安装好 Linux 时，
# 那么整个系统容量最大的应该是 /usr 。而 /proc 虽然有列出容量，但是那个容量是在内存中，
# 不占磁盘空间。至于 /proc 里头会列出一堆“No such file or directory” 的错误，
# 别担心！因为是内存内的程序，程序执行结束就会消失，因此会有些目录找不到，是正确的！
```

与 df 不一样的是，du 这个指令其实会直接到文件系统内去搜寻所有的文件数据， 所以上述第三个范例指令的运行会执行一小段时间！此外，在默认的情况下，容量的输出是以 KB 来设计的， 如果你想要知道目录占了多少 MB ，那么就使用 -m 这个参数即可啰！而， 如果你只想要知道该目录占了多少容量的话，使用 -s 就可以啦！

至于 -S 这个选项部分，由于 du 默认会将所有文件的大小均列出，因此假设你在 /etc 下面使用 du 时， 所有的文件大小，包括 /etc 下面的次目录容量也会被计算一次。然后最终的容量（/etc） 也会加总一次， 因此很多朋友都会误会 du 分析的结果不太对劲。所以啰，如果想要列出某目录下的全部数据， 或许也可以加上 -S 的选项，减少次目录的加总喔！

7.2.2 实体链接与符号链接： ln

关于链接（link）数据我们第五章的Linux文件属性及Linux文件种类与扩展名当中提过一些信息， 不过当时由于尚未讲到文件系统，因此无法较完整的介绍链接文件啦。不过在上一小节谈完了文件系统后， 我们可以来了解一下链接文件这玩意儿了。

在 Linux 下面的链接文件有两种，一种是类似 Windows 的捷径功能的文件，可以让你快速的链接到目标文件（或目录）； 另一种则是通过文件系统的 inode 链接来产生新文件名，而不是产生新文件！这种称为实体链接 （hard link）。 这两种玩意儿是完全不一样的东西呢！现在就分别来谈谈。

- Hard Link （实体链接, 硬式链接或实际链接）

在前一小节当中，我们知道几件重要的信息，包括：

- 每个文件都会占用一个 inode ，文件内容由 inode 的记录来指向；
- 想要读取该文件，必须要经过目录记录的文件名来指向到正确的 inode 号码才能读取。

也就是说，其实文件名只与目录有关，但是文件内容则与 inode 有关。那么想一想， 有没有可能有多个文件名对应到同一个 inode 号码呢？有的！那就是 hard link 的由来。 所以简单的说：hard link 只是在某个目录下新增一笔文件名链接到某 inode 号码的关连记录而已。

举个例子来说，假设我系统有个 /root/crontab 他是 /etc/crontab 的实体链接，也就是说这两个文件名链接到同一个 inode ， 自然这两个文件名的所有相关信息都会一模一样（除了文件名之外）。实际的情况可以如下所示：

```
[root@study ~]# ll -i /etc/crontab
34474855 -rw-r--r--. 1 root root 451 Jun 10  2014 /etc/crontab

[root@study ~]# ln /etc/crontab .   &lt;==创建实体链接的指令
[root@study ~]# ll -i /etc/crontab crontab
34474855 -rw-r--r--. 2 root root 451 Jun 10  2014 crontab
34474855 -rw-r--r--. 2 root root 451 Jun 10  2014 /etc/crontab
```

你可以发现两个文件名都链接到 34474855 这个 inode 号码，所以您瞧瞧，是否文件的权限/属性完全一样呢？ 因为这两个“文件名”其实是一模一样的“文件”啦！而且你也会发现第二个字段由原本的 1 变成 2 了！ 那个字段称为“链接”，这个字段的意义为：“有多少个文件名链接到这个 inode 号码”的意思。 如果将读取到正确数据的方式画成示意图，就类似如下画面：

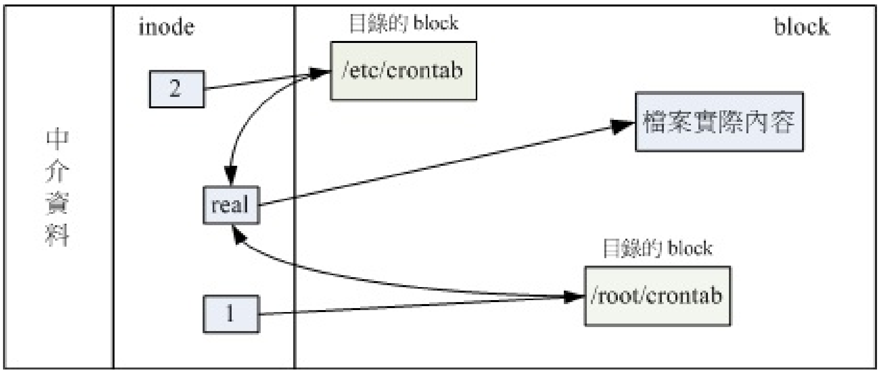


图7.2.1、实体链接的文件读取示意图

上图的意思是，你可以通过 1 或 2 的目录之 inode 指定的 block 找到两个不同的文件名，而不管使用哪个文件名均可以指到 real 那个 inode 去读取到最终数据！那这样有什么好处呢？最大的好处就是“安全”！如同上图中， 如果你将任何一个“文件名”删除，其实 inode 与 block 都还是存在的！ 此时你可以通过另一个“文件名”来读取到正确的文件数据喔！此外，不论你使用哪个“文件名”来编辑， 最终的结果都会写入到相同的 inode 与 block 中，因此均能进行数据的修改哩！

一般来说，使用 hard link 设置链接文件时，磁盘的空间与 inode 的数目都不会改变！ 我们还是由图 7.2.1 来看，由图中可以知道， hard link 只是在某个目录下的 block 多写入一个关连数据而已，既不会增加 inode 也不会耗用 block 数量哩！



Tips hard link 的制作中，其实还是可能会改变系统的 block 的，那就是当你新增这笔数据却刚好将目录的 block 填满时，就可能会新加一个 block 来记录文件名关连性，而导致磁盘空间的变化！不过，一般 hard link 所用掉的关连数据量很小，所以通常不会改变 inode 与磁盘空间的大小喔！

由图 7.2.1 其实我们也能够知道，事实上 hard link 应该仅能在单一文件系统中进行的，应该是不能够跨文件系统才对！ 因为图 7.2.1 就是在同一个 filesystem 上嘛！所以 hard link 是有限制的：

- 不能跨 Filesystem；
- 不能 link 目录。

不能跨 Filesystem 还好理解，那不能 hard link 到目录又是怎么回事呢？这是因为如果使用 hard link 链接到目录时， 链接的数据需要连同被链接目录下面的所有数据都创建链接，举例来说，如果你要将 /etc 使用实体链接创建一个 /etc_hd 的目录时，那么在 /etc_hd 下面的所有文件名同时都与 /etc 下面的文件名要创建 hard link 的，而不是仅链接到 /etc_hd 与 /etc 而

已。 并且，未来如果需要在 /etc_hd 下面创建新文件时，连带的， /etc 下面的数据又得要创建一次 hard link ，因此造成环境相当大的复杂度。 所以啰，目前 hard link 对于目录暂时还是不支持的啊！

- Symbolic Link （符号链接，亦即是捷径）

相对于 hard link ， Symbolic link 可就好理解多了，基本上， Symbolic link 就是在创建一个独立的文件，而这个文件会让数据的读取指向他 link 的那个文件的文件名！由于只是利用文件来做为指向的动作， 所以，当来源文件被删除之后，symbolic link 的文件会“开不了”， 会一直说“无法打开某文件！”。实际上就是找不到原始“文件名”而已啦！

举例来说，我们先创建一个符号链接文件链接到 /etc/crontab 去看看：

```
[root@study ~]# ln -s /etc/crontab crontab2
[root@study ~]# ll -i /etc/crontab /root/crontab2
34474855 -rw-r--r--. 2 root root 451 Jun 10  2014 /etc/crontab
53745909 lrwxrwxrwx. 1 root root  12 Jun 23 22:31 /root/crontab2 -&gt; /etc/crontab
```

由上表的结果我们可以知道两个文件指向不同的 inode 号码，当然就是两个独立的文件存在！ 而且链接文件的重要内容就是他会写上目标文件的“文件名”， 你可以发现为什么上表中链接文件的大小为 12 Bytes 呢？ 因为箭头（-->）右边的文件名“/etc/crontab”总共有 12 个英文，每个英文占用 1 个 Bytes ，所以文件大小就是 12Bytes了！

关于上述的说明，我们以如下图示来解释：

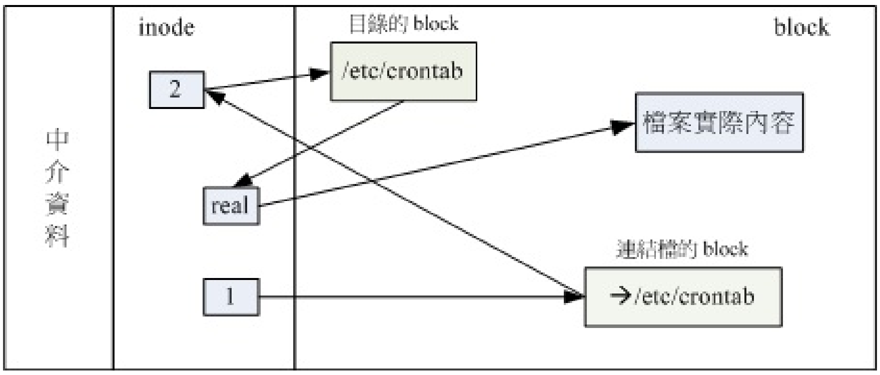


图7.2.2、符号链接的文件读取示意图

由 1 号 inode 读取到链接文件的内容仅有文件名，根据文件名链接到正确的目录去取得目标文件的 inode ， 最终就能够读取到正确的数据了。你可以发现的是，如果目标文件（/etc/crontab）被删除了，那么整个环节就会无法继续进行下去， 所以就会发生无法通过链接文件读取的问题了！

这里还是得特别留意，这个 Symbolic Link 与 Windows 的捷径可以给他划上等号，由 Symbolic link 所创建的文件为一个独立的新的文件，所以会占用掉 inode 与 block 喔！

由上面的说明来看，似乎 hard link 比较安全，因为即使某一个目录下的关连数据被杀掉了，也没有关系，只要有任何一个目录下存在着关连数据，那么该文件就不会不见！举上面的例子来说，我的 /etc/crontab 与 /root/crontab 指向同一个文件，如果我删除了 /etc/crontab 这个文件，该删除的动作其实只是将 /etc 目录下关于 crontab 的关连数据拿掉而已， crontab 所在的 inode 与 block 其实都没有被变动喔！

不过由于 Hard Link 的限制太多了，包括无法做“目录”的 link ， 所以在用途上面是比较受限的！反而是 Symbolic Link 的使用方面较广喔！好了， 说的天花乱坠，看你也差不多快要昏倒了！没关系，实作一下就知道怎么回事了！要制作链接文件就必须要使用 ln 这个指令呢！

```
[root@study ~]# ln [-sf] 来源文件 目标文件
选项与参数：
-s  ：如果不加任何参数就进行链接，那就是hard link，至于 -s 就是symbolic link
-f  ：如果 目标文件 存在时，就主动的将目标文件直接移除后再创建！

范例一：将 /etc/passwd 复制到 /tmp 下面，并且观察 inode 与 block
[root@study ~]# cd /tmp
[root@study tmp]# cp -a /etc/passwd .
[root@study tmp]# du -sb ; df -i .
6602    .   &lt;==先注意一下这里的容量是多少！
Filesystem                Inodes  IUsed    IFree IUse% Mounted on
/dev/mapper/centos-root 10485760 109748 10376012    2% /
# 利用 du 与 df 来检查一下目前的参数～那个 du -sb 是计算整个 /tmp 下面有多少 Bytes 的容量啦！

范例二：将 /tmp/passwd 制作 hard link 成为 passwd-hd 文件，并观察文件与容量
[root@study tmp]# ln passwd passwd-hd
[root@study tmp]# du -sb ; df -i .
6602    .
Filesystem                Inodes  IUsed    IFree IUse% Mounted on
/dev/mapper/centos-root 10485760 109748 10376012    2% /
# 仔细看，即使多了一个文件在 /tmp 下面，整个 inode 与 block 的容量并没有改变！

[root@study tmp]# ls -il passwd*
2668897 -rw-r--r--. 2 root root 2092 Jun 17 00:20 passwd
2668897 -rw-r--r--. 2 root root 2092 Jun 17 00:20 passwd-hd
# 原来是指向同一个 inode 啊！这是个重点啊！另外，那个第二栏的链接数也会增加！

范例三：将 /tmp/passwd 创建一个符号链接
[root@study tmp]# ln -s passwd passwd-so
[root@study tmp]# ls -li passwd*
2668897 -rw-r--r--. 2 root root 2092 Jun 17 00:20 passwd
2668897 -rw-r--r--. 2 root root 2092 Jun 17 00:20 passwd-hd
2668898 lrwxrwxrwx. 1 root root    6 Jun 23 22:40 passwd-so -&gt; passwd
# passwd-so 指向的 inode number 不同了！这是一个新的文件～这个文件的内容是指向
# passwd 的。passwd-so 的大小是 6Bytes ，因为 “passwd” 这个单字共有六个字符之故

[root@study tmp]# du -sb ; df -i .
6608    .
Filesystem                Inodes  IUsed    IFree IUse% Mounted on
/dev/mapper/centos-root 10485760 109749 10376011    2% /
# 呼呼！整个容量与 inode 使用数都改变啰～确实如此啊！

范例四：删除原始文件 passwd ，其他两个文件是否能够打开？
[root@study tmp]# rm passwd
[root@study tmp]# cat passwd-hd
.....（正常显示完毕！）
[root@study tmp]# cat passwd-so
cat: passwd-so: No such file or directory
[root@study tmp]# ll passwd*
-rw-r--r--. 1 root root 2092 Jun 17 00:20 passwd-hd
lrwxrwxrwx. 1 root root    6 Jun 23 22:40 passwd-so -&gt; passwd
# 怕了吧！符号链接果然无法打开！另外，如果符号链接的目标文件不存在，
# 其实文件名的部分就会有特殊的颜色显示喔！
```



Tips 还记得第五章当中，我们提到的 /tmp 这个目录是干嘛用的吗？是给大家作为暂存盘用的啊！ 所以，您会发现，过去我们在进行测试时，都会将数据移动到 /tmp 下面去练习～ 嘿嘿！因此，有事没事，记得将 /tmp 下面的一些怪异的数据清一清先！

要注意啰！使用 ln 如果不加任何参数的话，那么就是 Hard Link 啰！如同范例二的情况，增加了 hard link 之后，可以发现使用 ls -l 时，显示的 link 那一栏属性增加了！而如果这个时候砍掉 passwd 会发生什么事情呢？passwd-hd 的内容还是会跟原来 passwd 相同，但是 passwd-so 就会找不到该文件啦！

而如果 ln 使用 -s 的参数时，就做成差不多是 Windows 下面的“捷径”的意思。当你修改 Linux 下的 symbolic link 文件时，则更动的其实是“原始文件”， 所以不论你的这个原始文件被链接到哪里去，只要你修改了链接文件，原始文件就跟着变啰！ 以上面为例，由于你使用 -s 的参数创建一个名为 passwd-so 的文件，则你修改 passwd-so 时，其内容与 passwd 完全相同，并且，当你按下储存之后，被改变的将是 passwd 这个文件！

此外，如果你做了下面这样的链接：

> ln -s /bin /root/bin

那么如果你进入 /root/bin 这个目录下，“请注意呦！该目录其实是 /bin 这个目录，因为你做了链接文件了！”所以，如果你进入 /root/bin 这个刚刚创建的链接目录， 并且将其中的数据杀掉时，嗯！ /bin 里面的数据就通通不见了！这点请千万注意！所以赶紧利用“rm /root/bin ” 将这个链接文件删除吧！

基本上， Symbolic link 的用途比较广，所以您要特别留意 symbolic link 的用法呢！未来一定还会常常用到的啦！

- 关于目录的 link 数量：

或许您已经发现了，那就是，当我们以 hard link 进行“文件的链接”时，可以发现，在 ls -l 所显示的第二字段会增加一才对，那么请教，如果创建目录时，他默认的 link 数量会是多少？ 让我们来想一想，一个“空目录”里面至少会存在些什么？呵呵！就是存在 . 与 .. 这两个目录啊！ 那么，当我们创建一个新目录名称为 /tmp/testing 时，基本上会有三个东西，那就是：

- /tmp/testing
- /tmp/testing/.
- /tmp/testing/..

而其中 /tmp/testing 与 /tmp/testing/. 其实是一样的！都代表该目录啊～而 /tmp/testing/.. 则代表 /tmp 这个目录，所以说，当我们创建一个新的目录时，“新的目录的 link 数为 2 ，而上层目录的 link 数则会增加 1 ” 不信的话，我们来作个测试看看：

```
[root@study ~]# ls -ld /tmp
drwxrwxrwt. 14 root root 4096 Jun 23 22:42 /tmp
[root@study ~]# mkdir /tmp/testing1
[root@study ~]# ls -ld /tmp
drwxrwxrwt. 15 root root 4096 Jun 23 22:45 /tmp  # 这里的 link 数量加 1 了！
[root@study ~]# ls -ld /tmp/testing1
drwxr-xr-x. 2 root root 6 Jun 23 22:45 /tmp/testing1/
```

瞧！原本的所谓上层目录 /tmp 的 link 数量由 14 增加为 15 ，至于新目录 /tmp/testing 则为 2 ，这样可以理解目录的 link 数量的意义了吗？ ^_^

7.3 磁盘的分区、格式化、检验与挂载

对于一个系统管理者（ root ）而言，磁盘的的管理是相当重要的一环，尤其近来磁盘已经渐渐的被当成是消耗品了 ..... 如果我们想要在系统里面新增一颗磁盘时，应该有哪些动作需要做的呢：

1. 对磁盘进行分区，以创建可用的 partition ；
2. 对该 partition 进行格式化（format），以创建系统可用的 filesystem；
3. 若想要仔细一点，则可对刚刚创建好的 filesystem 进行检验；
4. 在 Linux 系统上，需要创建挂载点（亦即是目录），并将他挂载上来；

当然啰，在上述的过程当中，还有很多需要考虑的，例如磁盘分区（partition）需要定多大？ 是否需要加入 journal 的功能？inode 与 block 的数量应该如何规划等等的问题。但是这些问题的决定， 都需要与你的主机用途来加以考虑的～所以，在这个小节里面，鸟哥仅会介绍几个动作而已， 更详细的设置值，则需要以你未来的经验来参考啰！

7.3.1 观察磁盘分区状态

由于目前磁盘分区主要有 MBR 以及 GPT 两种格式，这两种格式所使用的分区工具不太一样！你当然可以使用本章预计最后才介绍的 parted 这个通通有支持的工具来处理，不过，我们还是比较习惯使用 fdisk 或者是 gdisk 来处理分区啊！因此，我们自然就得要去找一下目前系统有的磁盘有哪些？ 这些磁盘是 MBR 还是 GPT 等等的！这样才能处理啦！

- lsblk 列出系统上的所有磁盘列表

lsblk 可以看成“ list block device ”的缩写，就是列出所有储存设备的意思！这个工具软件真的很好用喔！来瞧一瞧！

```
[root@study ~]# lsblk [-dfimpt] [device]
选项与参数：
-d  ：仅列出磁盘本身，并不会列出该磁盘的分区数据
-f  ：同时列出该磁盘内的文件系统名称
-i  ：使用 ASCII 的线段输出，不要使用复杂的编码 （再某些环境下很有用）
-m  ：同时输出该设备在 /dev 下面的权限数据 （rwx 的数据）
-p  ：列出该设备的完整文件名！而不是仅列出最后的名字而已。
-t  ：列出该磁盘设备的详细数据，包括磁盘伫列机制、预读写的数据量大小等

范例一：列出本系统下的所有磁盘与磁盘内的分区信息
[root@study ~]# lsblk
NAME            MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
sr0              11:0    1 1024M  0 rom
vda             252:0    0   40G  0 disk             # 一整颗磁盘
&#124;-vda1          252:1    0    2M  0 part
&#124;-vda2          252:2    0    1G  0 part /boot
`-vda3          252:3    0   30G  0 part
  &#124;-centos-root 253:0    0   10G  0 lvm  /          # 在 vda3 内的其他文件系统
  &#124;-centos-swap 253:1    0    1G  0 lvm  [SWAP]
  `-centos-home 253:2    0    5G  0 lvm  /home
```

从上面的输出我们可以很清楚的看到，目前的系统主要有个 sr0 以及一个 vda 的设备，而 vda 的设备下面又有三个分区， 其中 vda3 甚至还有因为 LVM 产生的文件系统！相当的完整吧！从范例一我们来谈谈默认输出的信息有哪些。

- NAME：就是设备的文件名啰！会省略 /dev 等前导目录！
- MAJ:MIN：其实核心认识的设备都是通过这两个代码来熟悉的！分别是主要：次要设备代码！
- RM：是否为可卸载设备 （removable device），如光盘、USB 磁盘等等
- SIZE：当然就是容量啰！
- RO：是否为只读设备的意思
- TYPE：是磁盘 （disk）、分区 （partition） 还是只读存储器 （rom） 等输出
- MOUTPOINT：就是前一章谈到的挂载点！

```
范例二：仅列出 /dev/vda 设备内的所有数据的完整文件名
[root@study ~]# lsblk -ip /dev/vda
NAME                        MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
/dev/vda                    252:0    0   40G  0 disk
&#124;-/dev/vda1                    252:1    0    2M  0 part
&#124;-/dev/vda2                    252:2    0    1G  0 part /boot
`-/dev/vda3                 252:3    0   30G  0 part
  &#124;-/dev/mapper/centos-root 253:0    0   10G  0 lvm  /
  &#124;-/dev/mapper/centos-swap 253:1    0    1G  0 lvm  [SWAP]
  `-/dev/mapper/centos-home 253:2    0    5G  0 lvm  /home        # 完整的文件名，由 / 开始写
```

- blkid 列出设备的 UUID 等参数

虽然 lsblk 已经可以使用 -f 来列出文件系统与设备的 UUID 数据，不过，鸟哥还是比较习惯直接使用 blkid 来找出设备的 UUID 喔！ 什么是 UUID 呢？UUID 是全域单一识别码 （universally unique identifier），Linux 会将系统内所有的设备都给予一个独一无二的识别码， 这个识别码就可以拿来作为挂载或者是使用这个设备/文件系统之用了。

```
[root@study ~]# blkid
/dev/vda2: UUID="94ac5f77-cb8a-495e-a65b-2ef7442b837c" TYPE="xfs"
/dev/vda3: UUID="WStYq1-P93d-oShM-JNe3-KeDl-bBf6-RSmfae" TYPE="LVM2_member"
/dev/sda1: UUID="35BC-6D6B" TYPE="vfat"
/dev/mapper/centos-root: UUID="299bdc5b-de6d-486a-a0d2-375402aaab27" TYPE="xfs"
/dev/mapper/centos-swap: UUID="905dc471-6c10-4108-b376-a802edbd862d" TYPE="swap"
/dev/mapper/centos-home: UUID="29979bf1-4a28-48e0-be4a-66329bf727d9" TYPE="xfs"
```

如上所示，每一行代表一个文件系统，主要列出设备名称、UUID 名称以及文件系统的类型 （TYPE）！这对于管理员来说，相当有帮助！ 对于系统上面的文件系统观察来说，真是一目了然！

- parted 列出磁盘的分区表类型与分区信息

虽然我们已经知道了系统上面的所有设备，并且通过 blkid 也知道了所有的文件系统！不过，还是不清楚磁盘的分区类型。 这时我们可以通过简单的 parted 来输出喔！我们这里仅简单的利用他的输出而已～本章最后才会详细介绍这个指令的用法的！

```
[root@study ~]# parted device_name print

范例一：列出 /dev/vda 磁盘的相关数据
[root@study ~]# parted /dev/vda print
Model: Virtio Block Device (virtblk)         # 磁盘的模块名称(厂商)
Disk /dev/vda: 42.9GB                        # 磁盘的总容量
Sector size (logical/physical): 512B/512B    # 磁盘的每个逻辑/物理扇区容量
Partition Table: gpt                         # 分区表的格式 (MBR/GPT)
Disk Flags: pmbr_boot

Number  Start   End     Size    File system  Name  Flags     # 下面才是分区数据
 1      1049kB  3146kB  2097kB                     bios_grub
 2      3146kB  1077MB  1074MB  xfs
 3      1077MB  33.3GB  32.2GB                     lvm
```

看到上表的说明，你就知道啦！我们用的就是 GPT 的分区格式喔！这样会观察磁盘分区了吗？接下来要来操作磁盘分区了喔！

7.3.2 磁盘分区： gdisk/fdisk

接下来我们想要进行磁盘分区啰！要注意的是：“MBR 分区表请使用 fdisk 分区， GPT 分区表请使用 gdisk 分区！” 这个不要搞错～否则会分区失败的！另外，这两个工具软件的操作很类似，执行了该软件后，可以通过该软件内部的说明数据来操作， 因此不需要硬背！只要知道方法即可。刚刚从上面 parted 的输出结果，我们也知道鸟哥这个测试机使用的是 GPT 分区， 因此下面通通得要使用 gdisk 来分区才行！

- gdisk

```
[root@study ~]# gdisk 设备名称

范例：由前一小节的 lsblk 输出，我们知道系统有个 /dev/vda，请观察该磁盘的分区与相关数据
[root@study ~]# gdisk /dev/vda  &lt;==仔细看，不要加上数字喔！
GPT fdisk （gdisk） version 0.8.6

Partition table scan:
  MBR: protective
  BSD: not present
  APM: not present
  GPT: present

Found valid GPT with protective MBR; using GPT.  &lt;==找到了 GPT 的分区表！

Command （? for help）：     &lt;==这里可以让你输入指令动作，可以按问号 （?） 来查看可用指令
Command （? for help）：?
b       back up GPT data to a file
c       change a partition's name
d       delete a partition           # 删除一个分区
i       show detailed information on a partition
l       list known partition types
n       add a new partition          # 增加一个分区
o       create a new empty GUID partition table （GPT）
p       print the partition table    # 印出分区表 （常用）
q       quit without saving changes  # 不储存分区就直接离开 gdisk
r       recovery and transformation options （experts only）
s       sort partitions
t       change a partition's type code
v       verify disk
w       write table to disk and exit # 储存分区操作后离开 gdisk
x       extra functionality （experts only）
?       print this menu
Command （? for help）：
```

你应该要通过 lsblk 或 blkid 先找到磁盘，再用 parted /dev/xxx print 来找出内部的分区表类型，之后才用 gdisk 或 fdisk 来操作系统。 上表中可以发现 gdisk 会扫描 MBR 与 GPT 分区表，不过这个软件还是单纯使用在 GPT 分区表比较好啦！

老实说，使用 gdisk 这支程序是完全不需要背指令的！如同上面的表格中，你只要按下 ? 就能够看到所有的动作！ 比较重要的动作在上面已经用底线画出来了，你可以参考看看。其中比较不一样的是“q 与 w”这两个玩意儿！ 不管你进行了什么动作，只要离开 gdisk 时按下“q”，那么所有的动作“都不会生效！”相反的， 按下“w”就是动作生效的意思。所以，你可以随便玩 gdisk ，只要离开时按下的是“q”即可。 ^_^！ 好了，先来看看分区表信息吧！

```
Command （? for help）：p  &lt;== 这里可以输出目前磁盘的状态
Disk /dev/vda: 83886080 sectors, 40.0 GiB                     # 磁盘文件名/扇区数与总容量
Logical sector size: 512 Bytes                                # 单一扇区大小为 512 Bytes
Disk identifier （GUID）: A4C3C813-62AF-4BFE-BAC9-112EBD87A483  # 磁盘的 GPT 识别码
Partition table holds up to 128 entries
First usable sector is 34, last usable sector is 83886046
Partitions will be aligned on 2048-sector boundaries
Total free space is 18862013 sectors （9.0 GiB）

Number  Start （sector）    End （sector）  Size       Code  Name # 下面为完整的分区信息了！
   1            2048            6143   2.0 MiB     EF02        # 第一个分区数据
   2            6144         2103295   1024.0 MiB  0700
   3         2103296        65026047   30.0 GiB    8E00
# 分区编号 开始扇区号码  结束扇区号码  容量大小
Command （? for help）：q
# 想要不储存离开吗？按下 q 就对了！不要随便按 w 啊！
```

使用“ p ”可以列出目前这颗磁盘的分区表信息，这个信息的上半部在显示整体磁盘的状态。以鸟哥这颗磁盘为例，这个磁盘共有 40GB 左右的容量，共有 83886080 个扇区，每个扇区的容量为 512Bytes。 要注意的是，现在的分区主要是以扇区为最小的单位喔！

下半部的分区表信息主要在列出每个分区的个别信息项目。每个项目的意义为：

- Number：分区编号，1 号指的是 /dev/vda1 这样计算。
- Start （sector）：每一个分区的开始扇区号码位置
- End （sector）：每一个分区的结束扇区号码位置，与 start 之间可以算出分区的总容量
- Size：就是分区的容量了
- Code：在分区内的可能的文件系统类型。Linux 为 8300，swap 为 8200。不过这个项目只是一个提示而已，不见得真的代表此分区内的文件系统喔！
- Name：文件系统的名称等等。

从上表我们可以发现几件事情：

- 整部磁盘还可以进行额外的分区，因为最大扇区为 83886080，但只使用到 65026047 号而已；
- 分区的设计中，新分区通常选用上一个分区的结束扇区号码数加 1 作为起始扇区号码！

这个 gdisk 只有 root 才能执行，此外，请注意，使用的“设备文件名”请不要加上数字，因为 partition 是针对“整个磁盘设备”而不是某个 partition 呢！所以执行“ gdisk /dev/vda1 ” 就会发生错误啦！要使用 gdisk /dev/vda 才对！



Tips 再次强调，你可以使用 gdisk 在您的磁盘上面胡搞瞎搞的进行实际操作，都不打紧，但是请“千万记住，不要按下 w 即可！”离开的时候按下 q 就万事无妨啰！ 此外，不要在 MBR 分区上面使用 gdisk，因为如果指令按错，恐怕你的分区纪录会全部死光光！也不要在 GPT 上面使用 fdisk 啦！切记切记！

- 用 gdisk 新增分区

如果你是按照鸟哥建议的方式去安装你的 CentOS 7，那么你的磁盘应该会预留一块容量来做练习的。如果没有的话， 那么你可能需要找另外一颗磁盘来让你练习才行呦！而经过上面的观察，我们也确认系统还有剩下的容量可以来操作练习分区！ 假设我需要有如下的分区需求：

- 1GB 的 xfs 文件系统 （Linux）
- 1GB 的 vfat 文件系统 （Windows）
- 0.5GB 的 swap （Linux swap）（这个分区等一下会被删除喔！）

那就来处理处理！

```
[root@study ~]# gdisk /dev/vda
Command （? for help）: p
Number  Start （sector）    End （sector）  Size       Code  Name
   1          2048            6143   2.0 MiB     EF02
   2          6144         2103295   1024.0 MiB  0700
   3       2103296        65026047   30.0 GiB    8E00
# 找出最后一个 sector 的号码是很重要的！

Command （? for help）: ?  # 查一下增加分区的指令为何
Command （? for help）: n  # 就是这个！所以开始新增的行为！
Partition number （4-128, default 4）: 4  # 默认就是 4 号，所以也能 enter 即可！
First sector （34-83886046, default = 65026048） or {+-}size{KMGTP}: 65026048  # 也能 ente
Last sector （65026048-83886046, default = 83886046） or {+-}size{KMGTP}: +1G  # 决不要 ent
# 这个地方可有趣了！我们不需要自己去计算扇区号码，通过 +容量 的这个方式，
# 就可以让 gdisk 主动去帮你算出最接近你需要的容量的扇区号码喔！

Current type is 'Linux filesystem'
Hex code or GUID （L to show codes, Enter = 8300）: # 使用默认值即可～直接 enter 下去！
# 这里在让你选择未来这个分区预计使用的文件系统！默认都是 Linux 文件系统的 8300 啰！

Command （? for help）: p
Number  Start （sector）    End （sector）  Size       Code  Name
   1          2048            6143   2.0 MiB     EF02
   2          6144         2103295   1024.0 MiB  0700
   3       2103296        65026047   30.0 GiB    8E00
 4        65026048        67123199   1024.0 MiB  8300  Linux filesystem
```

重点在“ Last sector ”那一行，那行绝对不要使用默认值！因为默认值会将所有的容量用光！因此它默认选择最大的扇区号码！ 因为我们仅要 1GB 而已，所以你得要加上 +1G 这样即可！不需要计算 sector 的数量，gdisk 会根据你填写的数值， 直接计算出最接近该容量的扇区数！每次新增完毕后，请立即“ p ”查看一下结果喔！请继续处理后续的两个分区！ 最终出现的画面会有点像下面这样才对！

```
Command （? for help）: p
Number  Start （sector）    End （sector）  Size       Code  Name
   1          2048            6143   2.0 MiB     EF02
   2          6144         2103295   1024.0 MiB  0700
   3       2103296        65026047   30.0 GiB    8E00
   4      65026048        67123199   1024.0 MiB  8300  Linux filesystem
   5      67123200        69220351   1024.0 MiB  0700  Microsoft basic data
   6      69220352        70244351   500.0 MiB   8200  Linux swap
```

基本上，几乎都用默认值，然后通过 +1G, +500M 来创建所需要的另外两个分区！比较有趣的是文件系统的 ID 啦！一般来说， Linux 大概都是 8200/8300/8e00 等三种格式， Windows 几乎都用 0700 这样，如果忘记这些数字，可以在 gdisk 内按下：“ L ”来显示喔！ 如果一切的分区状态都正常的话，那么就直接写入磁盘分区表吧！

```
Command （? for help）: w

Final checks complete. About to write GPT data. THIS WILL OVERWRITE EXISTING
PARTITIONS!!

Do you want to proceed? （Y/N）: y
OK; writing new GUID partition table （GPT） to /dev/vda.
Warning: The kernel is still using the old partition table.
The new table will be used at the next reboot.
The operation has completed successfully.
# gdisk 会先警告你可能的问题，我们确定分区是对的，这时才按下 y ！不过怎么还有警告？
# 这是因为这颗磁盘目前正在使用当中，因此系统无法立即载入新的分区表～

[root@study ~]# cat /proc/partitions
major minor  #blocks  name

 252        0   41943040 vda
 252        1       2048 vda1
 252        2    1048576 vda2
 252        3   31461376 vda3
 253        0   10485760 dm-0
 253        1    1048576 dm-1
 253        2    5242880 dm-2
# 你可以发现，并没有 vda4, vda5, vda6 喔！因为核心还没有更新！
```

因为 Linux 此时还在使用这颗磁盘，为了担心系统出问题，所以分区表并没有被更新喔！这个时候我们有两个方式可以来处理！ 其中一个是重新开机，不过很讨厌！另外一个则是通过 partprobe 这个指令来处理即可！

- partprobe 更新 Linux 核心的分区表信息

```
[root@study ~]# partprobe [-s]  # 你可以不要加 -s ！那么屏幕不会出现讯息！
[root@study ~]# partprobe -s    # 不过还是建议加上 -s 比较清晰！
/dev/vda: gpt partitions 1 2 3 4 5 6

[root@study ~]# lsblk /dev/vda  # 实际的磁盘分区状态
NAME            MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda             252:0    0   40G  0 disk
&#124;-vda1          252:1    0    2M  0 part
&#124;-vda2          252:2    0    1G  0 part /boot
&#124;-vda3          252:3    0   30G  0 part
&#124; &#124;-centos-root 253:0    0   10G  0 lvm  /
&#124; &#124;-centos-swap 253:1    0    1G  0 lvm  [SWAP]
&#124; `-centos-home 253:2    0    5G  0 lvm  /home
&#124;-vda4          252:4    0    1G  0 part
&#124;-vda5          252:5    0    1G  0 part
`-vda6          252:6    0  500M  0 part

[root@study ~]# cat /proc/partitions  # 核心的分区纪录
major minor  #blocks  name

 252        0   41943040 vda
 252        1       2048 vda1
 252        2    1048576 vda2
 252        3   31461376 vda3
 252        4    1048576 vda4
 252        5    1048576 vda5
 252        6     512000 vda6
# 现在核心也正确的抓到了分区参数了！
```

- 用 gdisk 删除一个分区

已经学会了新增分区，那么删除分区呢？好！现在让我们将刚刚创建的 /dev/vda6 删除！你该如何进行呢？鸟哥下面很快的处理一遍， 大家赶紧来瞧一瞧先！

```
[root@study ~]# gdisk /dev/vda
Command （? for help）: p
Number  Start （sector）    End （sector）  Size       Code  Name
   1            2048            6143   2.0 MiB     EF02
   2            6144         2103295   1024.0 MiB  0700
   3         2103296        65026047   30.0 GiB    8E00
   4        65026048        67123199   1024.0 MiB  8300  Linux filesystem
   5        67123200        69220351   1024.0 MiB  0700  Microsoft basic data
   6        69220352        70244351   500.0 MiB   8200  Linux swap

Command （? for help）: d
Partition number （1-6）: 6

Command （? for help）: p
# 你会发现 /dev/vda6 不见了！非常棒！没问题就写入吧！

Command （? for help）: w
# 同样会有一堆讯息！鸟哥就不重复输出了！自己选择 y 来处理吧！

[root@study ~]# lsblk
# 你会发现！怪了！怎么还是有 /dev/vda6 呢？没办法！还没有更新核心的分区表啊！所以当然有错！

[root@study ~]# partprobe -s
[root@study ~]# lsblk
# 这个时候，那个 /dev/vda6 才真的消失不见了！了解吧！
```



Tips 万分注意！不要去处理一个正在使用中的分区！例如，我们的系统现在已经使用了 /dev/vda2 ，那如果你要删除 /dev/vda2 的话， 必须要先将 /dev/vda2 卸载，否则直接删除该分区的话，虽然磁盘还是慧写入正确的分区信息，但是核心会无法更新分区表的信息的！ 另外，文件系统与 Linux 系统的稳定性，恐怕也会变得怪怪的！反正！千万不要处理正在使用中的文件系统就对了！

- fdisk

虽然 MBR 分区表在未来应该会慢慢的被淘汰，毕竟现在磁盘容量随便都大于 2T 以上了。而对于在 CentOS 7.x 中还无法完整支持 GPT 的 fdisk 来说， 这家伙真的英雄无用武之地了啦！不过依旧有些旧的系统，以及虚拟机的使用上面，还是有小磁盘存在的空间！这时处理 MBR 分区表， 就得要使用 fdisk 啰！

因为 fdisk 跟 gdisk 使用的方式几乎一样！只是一个使用 ? 作为指令提示数据，一个使用 m 作为提示这样而已。 此外，fdisk 有时会使用柱面 （cylinder） 作为分区的最小单位，与 gdisk 默认使用 sector 不太一样！大致上只是这点差别！ 另外， MBR 分区是有限制的（Primary, Extended, Logical...）！不要忘记了！鸟哥这里不使用范例了，毕竟示范机上面也没有 MBR 分区表... 这里仅列出相关的指令给大家对照参考啰！

```
[root@study ~]# fdisk /dev/sda
Command （m for help）: m   &lt;== 输入 m 后，就会看到下面这些指令介绍
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition            &lt;==删除一个partition
   l   list known partition types
   m   print this menu
   n   add a new partition           &lt;==新增一个partition
   o   create a new empty DOS partition table
   p   print the partition table     &lt;==在屏幕上显示分区表
   q   quit without saving changes   &lt;==不储存离开fdisk程序
   s   create a new empty Sun disklabel
   t   change a partition's system id
   u   change display/entry units
   v   verify the partition table
   w   write table to disk and exit  &lt;==将刚刚的动作写入分区表
   x   extra functionality （experts only）
```

7.3.3 磁盘格式化（创建文件系统）

分区完毕后自然就是要进行文件系统的格式化啰！格式化的指令非常的简单，那就是“make filesystem, mkfs” 这个指令啦！这个指令其实是个综合的指令，他会去调用正确的文件系统格式化工具软件！因为 CentOS 7 使用 xfs 作为默认文件系统， 下面我们会先介绍 mkfs.xfs ，之后介绍新一代的 EXT 家族成员 mkfs.ext4，最后再聊一聊 mkfs 这个综合指令吧！

- XFS 文件系统 mkfs.xfs

我们常听到的“格式化”其实应该称为“创建文件系统 （make filesystem）”才对啦！所以使用的指令是 mkfs 喔！那我们要创建的其实是 xfs 文件系统， 因此使用的是 mkfs.xfs 这个指令才对。这个指令是这样使用的：

```
[root@study ~]# mkfs.xfs [-b bsize] [-d parms] [-i parms] [-l parms] [-L label] [-f] \
                         [-r parms] 设备名称
选项与参数：
关於单位：下面只要谈到“数值”时，没有加单位则为 Bytes 值，可以用 k,m,g,t,p （小写）等来解释
          比较特殊的是 s 这个单位，它指的是 sector 的“个数”喔！
-b  ：后面接的是 block 容量，可由 512 到 64k，不过最大容量限制为 Linux 的 4k 喔！
-d  ：后面接的是重要的 data section 的相关参数值，主要的值有：
      agcount=数值   ：设置需要几个储存群组的意思（AG），通常与 CPU 有关
      agsize=数值    ：每个 AG 设置为多少容量的意思，通常 agcount/agsize 只选一个设置即可
      file           ：指的是“格式化的设备是个文件而不是个设备”的意思！（例如虚拟磁盘）
      size=数值      ：data section 的容量，亦即你可以不将全部的设备容量用完的意思
      su=数值        ：当有 RAID 时，那个 stripe 数值的意思，与下面的 sw 搭配使用
      sw=数值        ：当有 RAID 时，用于储存数据的磁盘数量（须扣除备份碟与备用碟）
      sunit=数值     ：与 su 相当，不过单位使用的是“几个 sector（512Bytes大小）”的意思
      swidth=数值    ：就是 su*sw 的数值，但是以“几个 sector（512Bytes大小）”来设置
-f  ：如果设备内已经有文件系统，则需要使用这个 -f 来强制格式化才行！
-i  ：与 inode 有较相关的设置，主要的设置值有：
      size=数值      ：最小是 256Bytes 最大是 2k，一般保留 256 就足够使用了！
      internal=[0&#124;1]：log 设备是否为内置？默认为 1 内置，如果要用外部设备，使用下面设置
      logdev=device  ：log 设备为后面接的那个设备上头的意思，需设置 internal=0 才可！
      size=数值      ：指定这块登录区的容量，通常最小得要有 512 个 block，大约 2M 以上才行！
-L  ：后面接这个文件系统的标头名称 Label name 的意思！
-r  ：指定 realtime section 的相关设置值，常见的有：
      extsize=数值   ：就是那个重要的 extent 数值，一般不须设置，但有 RAID 时，
                      最好设置与 swidth 的数值相同较佳！最小为 4K 最大为 1G 。

范例：将前一小节分区出来的 /dev/vda4 格式化为 xfs 文件系统
[root@study ~]# mkfs.xfs /dev/vda4
meta-data=/dev/vda4        isize=256    agcount=4, agsize=65536 blks
         =                 sectsz=512   attr=2, projid32bit=1
         =                 crc=0        finobt=0
data     =                 bsize=4096   blocks=262144, imaxpct=25
         =                 sunit=0      swidth=0 blks
naming   =version 2        bsize=4096   ascii-ci=0 ftype=0
log      =internal log     bsize=4096   blocks=2560, version=2
         =                 sectsz=512   sunit=0 blks, lazy-count=1
realtime =none             extsz=4096   blocks=0, rtextents=0
# 很快格是化完毕！都用默认值！较重要的是 inode 与 block 的数值

[root@study ~]# blkid /dev/vda4
/dev/vda4: UUID="39293f4f-627b-4dfd-a015-08340537709c" TYPE="xfs"
# 确定创建好 xfs 文件系统了！
```

使用默认的 xfs 文件系统参数来创建系统即可！速度非常快！如果我们有其他额外想要处理的项目，才需要加上一堆设置值！举例来说，因为 xfs 可以使用多个数据流来读写系统，以增加速度，因此那个 agcount 可以跟 CPU 的核心数来做搭配！举例来说，如果我的服务器仅有一颗 4 核心，但是有启动 Intel 超线程功能，则系统会仿真出 8 颗 CPU 时，那个 agcount 就可以设置为 8 喔！举个例子来瞧瞧：

```
范例：找出你系统的 CPU 数，并据以设置你的 agcount 数值
[root@study ~]# grep 'processor' /proc/cpuinfo
processor       : 0
processor       : 1
# 所以就是有两颗 CPU 的意思，那就来设置设置我们的 xfs 文件系统格式化参数吧！！

[root@study ~]# mkfs.xfs -f -d agcount=2 /dev/vda4
meta-data=/dev/vda4        isize=256    agcount=2, agsize=131072 blks
         =                 sectsz=512   attr=2, projid32bit=1
         =                 crc=0        finobt=0
.....（下面省略）.....
# 可以跟前一个范例对照看看，可以发现 agcount 变成 2 了喔！
# 此外，因为已经格式化过一次，因此 mkfs.xfs 可能会出现不给你格式化的警告！因此需要使用 -f
```

- XFS 文件系统 for RAID 性能优化 （Optional）

我们在第14章会持续谈到进阶文件系统的设置，其中就有磁盘阵列这个东西！磁盘阵列是多颗磁盘组成一颗大磁盘的意思， 利用同步写入到这些磁盘的技术，不但可以加快读写速度，还可以让某一颗磁盘坏掉时，整个文件系统还是可以持续运行的状态！那就是所谓的容错。

基本上，磁盘阵列 （RAID） 就是通过将文件先细分为数个小型的分区区块 （stripe） 之后，然后将众多的 stripes 分别放到磁盘阵列里面的所有磁盘， 所以一个文件是被同时写入到多个磁盘去，当然性能会好一些。为了文件的保全性，所以在这些磁盘里面，会保留数个（与磁盘阵列的规划有关） 备份磁盘 （parity disk） ， 以及可能会保留一个以上的备用磁盘（spare disk） ，这些区块基本上会占用掉磁盘阵列的总容量，不过对于数据的保全会比较有保障！

那个分区区块 stripe 的数值大多介于 4K 到 1M 之间，这与你的磁盘阵列卡支持的项目有关。stripe 与你的文件数据容量以及性能相关性较高。 当你的系统大多是大型文件时，一般建议 stripe 可以设置大一些，这样磁盘阵列读/写的频率会降低，性能会提升。如果是用于系统，那么小文件比较多的情况下， stripe 建议大约在 64K 左右可能会有较佳的性能。不过，还是都须要经过测试啦！完全是 case by case 的情况。 更多详细的磁盘阵列我们在第 14 章再来谈，这里先有个大概的认识即可。14 章看完之后，再回来这个小节瞧瞧啰！

文件系统的读写要能够有最优化，最好能够搭配磁盘阵列的参数来设计，这样性能才能够起来！也就是说，你可以先在文件系统就将 stripe 规划好， 那交给 RAID 去存取时，它就无须重复进行文件的 stripe 过程，性能当然会更好！那格式化时，最优化性能与什么咚咚有关呢？我们来假设个环境好了：

- 我有两个线程的 CPU 数量，所以 agcount 最好指定为 2
- 当初设置 RAID 的 stripe 指定为 256K 这么大，因此 su 最好设置为 256k
- 设置的磁盘阵列有 8 颗，因为是 RAID5 的设置，所以有一个 parity （备份碟） ，因此指定 sw 为 7
- 由上述的数据中，我们可以发现数据宽度 （swidth） 应该就是 256K*7 得到 1792K，可以指定 extsize 为 1792k

相关数据的来源可以参考文末[7]的说明，这里仅快速的使用 mkfs.xfs 的参数来处理格式化的动作喔！

```
[root@study ~]# mkfs.xfs -f -d agcount=2,su=256k,sw=7 -r extsize=1792k /dev/vda4
meta-data=/dev/vda4              isize=256    agcount=2, agsize=131072 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=0        finobt=0
data     =                       bsize=4096   blocks=262144, imaxpct=25
         =                       sunit=64     swidth=448 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=0
log      =internal log           bsize=4096   blocks=2560, version=2
         =                       sectsz=512   sunit=64 blks, lazy-count=1
realtime =none                   extsz=1835008 blocks=0, rtextents=0
```

从输出的结果来看， agcount 没啥问题， sunit 结果是 64 个 block，因为每个 block 为 4K，所以算出来容量就是 256K 也没错！ 那个 swidth 也相同！使用 448 * 4K 得到 1792K！那个 extsz 则是算成 Bytes 的单位，换算结果也没错啦！上面是个方式，那如果使用 sunit 与 swidth 直接套用在 mkfs.xfs 当中呢？那你得小心了！因为指令中的这两个参数用的是“几个 512Bytes 的 sector 数量”的意思！ 是“数量”单位而不是“容量”单位！因此先计算为：

- sunit = 256K/512Byte*1024（Bytes/K） = 512 个 sector
- swidth = 7 个磁盘 *sunit* = 7 512 = 3584 个 sector

所以指令就得要变成如下模样：

```
[root@study ~]# mkfs.xfs -f -d agcount=2,sunit=512,swidth=3584 -r extsize=1792k /dev/vda4
```

再说一次，这边你大概先有个概念即可，看不懂也没关系！等到 14 章看完后，未来回到这里，应该就能够看得懂了！ 多看几次！多做几次～操作系统的练习就是这样才能学的会！看得懂！ ^_^

- EXT4 文件系统 mkfs.ext4

如果想要格式化为 ext4 的传统 Linux 文件系统的话，可以使用 mkfs.ext4 这个指令即可！这个指令的参数快速的介绍一下！

```
[root@study ~]# mkfs.ext4 [-b size] [-L label] 设备名称
选项与参数：
-b  ：设置 block 的大小，有 1K, 2K, 4K 的容量，
-L  ：后面接这个设备的标头名称。

范例：将 /dev/vda5 格式化为 ext4 文件系统
[root@study ~]# mkfs.ext4 /dev/vda5
mke2fs 1.42.9 (28-Dec-2013)
Filesystem label=                                  # 显示 Label name
OS type: Linux
Block size=4096 (log=2)                             # 每一个 block 的大小
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks             # 跟 RAID 相关性较高
65536 inodes, 262144 blocks                        # 总计 inode/block 的数量
13107 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=268435456
8 block groups                                     # 共有 8 个 block groups 喔！
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

[root@study ~]# dumpe2fs -h /dev/vda5
dumpe2fs 1.42.9 (28-Dec-2013)
Filesystem volume name:   &lt;none&gt;
Last mounted on:          &lt;not available&gt;
Filesystem UUID:          3fd5cc6f-a47d-46c0-98c0-d43b072e0e12
....(中间省略)....
Inode count:              65536
Block count:              262144
Block size:               4096
Blocks per group:         32768
Inode size:               256
Journal size:             32M
```

由于数据量较大，因此鸟哥仅列出比较重要的项目而已，提供给你参考。另外，本章稍早之前介绍的 dumpe2fs 现在也可以测试练习了！查阅一下相关的数据吧！ 因为 ext4 的默认值已经相当适合我们系统使用，大部分的默认值写入于我们系统的 /etc/mke2fs.conf 这个文件中，有兴趣可以自行前往查阅。 也因此，我们无须额外指定 inode 的容量，系统都帮我们做好默认值啰！只需要得到 uuid 这个咚咚即可啦！

- 其他文件系统 mkfs

mkfs 其实是个综合指令而已，当我们使用 mkfs -t xfs 时，它就会跑去找 mkfs.xfs 相关的参数给我们使用！ 如果想要知道系统还支持哪种文件系统的格式化功能，直接按 [tabl] 就很清楚了！

```
[root@study ~]# mkfs[tab][tab]
mkfs         mkfs.btrfs   mkfs.cramfs  mkfs.ext2    mkfs.ext3    mkfs.ext4
mkfs.fat     mkfs.minix   mkfs.msdos   mkfs.vfat    mkfs.xfs
```

所以系统还有支持 ext2/ext3/vfat 等等多种常用的文件系统喔！那如果要将刚刚的 /dev/vda5 重新格式化为 VFAT 文件系统呢？

```
[root@study ~]# mkfs -t vfat /dev/vda5
[root@study ~]# blkid /dev/vda5
/dev/vda5: UUID="7130-6012" TYPE="vfat" PARTLABEL="Microsoft basic data"

[root@study ~]# mkfs.ext4 /dev/vda5
[root@study ~]# blkid /dev/vda4 /dev/vda5
/dev/vda4: UUID="e0a6af55-26e7-4cb7-a515-826a8bd29e90" TYPE="xfs"
/dev/vda5: UUID="899b755b-1da4-4d1d-9b1c-f762adb798e1" TYPE="ext4"
```

上面就是我们这个章节最后的结果了！ /dev/vda4 是 xfs 文件系统，而 /dev/vda5 是 ext4 文件系统喔！都有练习妥当了嘛？



Tips 越来越多同学上课都不听讲，只是很单纯的将鸟哥在屏幕操作的过程“拍照”下来而已～当鸟哥说“开始操作！等一下要检查喔！” 大家就拼命的从手机里面将刚刚的照片抓出来，一个一个指令照着打～

不过，屏幕并不能告诉你“ [tab] 按钮其实不是按下 enter”的结果， 如上所示，同学拼命的按下 mkfs 之后，却没有办法得到下面出现的众多指令，就开始举手...老师！我没办法作到你讲的画面...

拜托读者们，请注意：“我们是要练习 Linux 系统，不是要练习 "英文打字"”啦！英文打字回家练就好了！ @_@

7.3.4 文件系统检验

由于系统在运行时谁也说不准啥时硬件或者是电源会有问题，所以“死机”可能是难免的情况（不管是硬件还是软件）。 现在我们知道文件系统运行时会有磁盘与内存数据非同步的状况发生，因此莫名其妙的死机非常可能导致文件系统的错乱。 问题来啦，如果文件系统真的发生错乱的话，那该如何是好？就...挽救啊！不同的文件系统救援的指令不太一样，我们主要针对 xfs 及 ext4 这两个主流来说明而已喔！

- xfs_repair 处理 XFS 文件系统

当有 xfs 文件系统错乱才需要使用这个指令！所以，这个指令最好是不要用到啦！但有问题发生时，这个指令却又很重要...

```
[root@study ~]# xfs_repair [-fnd] 设备名称
选项与参数：
-f  ：后面的设备其实是个文件而不是实体设备
-n  ：单纯检查并不修改文件系统的任何数据 （检查而已）
-d  ：通常用在单人维护模式下面，针对根目录 （/） 进行检查与修复的动作！很危险！不要随便使用

范例：检查一下刚刚创建的 /dev/vda4 文件系统
[root@study ~]# xfs_repair /dev/vda4
Phase 1 - find and verify superblock...
Phase 2 - using internal log
Phase 3 - for each AG...
Phase 4 - check for duplicate blocks...
Phase 5 - rebuild AG headers and trees...
Phase 6 - check inode connectivity...
Phase 7 - verify and correct link counts...
done
# 共有 7 个重要的检查流程！详细的流程介绍可以 man xfs_repair 即可！

范例：检查一下系统原本就有的 /dev/centos/home 文件系统
[root@study ~]# xfs_repair /dev/centos/home
xfs_repair: /dev/centos/home contains a mounted filesystem
xfs_repair: /dev/centos/home contains a mounted and writable filesystem

fatal error -- couldn't initialize XFS library
```

xfs_repair 可以检查/修复文件系统，不过，因为修复文件系统是个很庞大的任务！因此，修复时该文件系统不能被挂载！ 所以，检查与修复 /dev/vda4 没啥问题，但是修复 /dev/centos/home 这个已经挂载的文件系统时，嘿嘿！就出现上述的问题了！ 没关系，若可以卸载，卸载后再处理即可。

Linux 系统有个设备无法被卸载，那就是根目录啊！如果你的根目录有问题怎办？这时得要进入单人维护或救援模式，然后通过 -d 这个选项来处理！ 加入 -d 这个选项后，系统会强制检验该设备，检验完毕后就会自动重新开机啰！不过，鸟哥完全不打算要进行这个指令的实做... 永远都不希望实做这东西...

- fsck.ext4 处理 EXT4 文件系统

fsck 是个综合指令，如果是针对 ext4 的话，建议直接使用 fsck.ext4 来检测比较妥当！那 fsck.ext4 的选项有下面几个常见的项目：

```
[root@study ~]# fsck.ext4 [-pf] [-b superblock] 设备名称
选项与参数：
-p  ：当文件系统在修复时，若有需要回复 y 的动作时，自动回复 y 来继续进行修复动作。
-f  ：强制检查！一般来说，如果 fsck 没有发现任何 unclean 的旗标，不会主动进入
      细部检查的，如果您想要强制 fsck 进入细部检查，就得加上 -f 旗标啰！
-D  ：针对文件系统下的目录进行最优化配置。
-b  ：后面接 superblock 的位置！一般来说这个选项用不到。但是如果你的 superblock 因故损毁时，
      通过这个参数即可利用文件系统内备份的 superblock 来尝试救援。一般来说，superblock 备份在：
      1K block 放在 8193, 2K block 放在 16384, 4K block 放在 32768

范例：找出刚刚创建的 /dev/vda5 的另一块 superblock，并据以检测系统
[root@study ~]# dumpe2fs -h /dev/vda5 &#124; grep 'Blocks per group'
Blocks per group:         32768
# 看起来每个 block 群组会有 32768 个 block，因此第二个 superblock 应该就在 32768 上！
# 因为 block 号码为 0 号开始编的！

[root@study ~]# fsck.ext4 -b 32768 /dev/vda5
e2fsck 1.42.9 （28-Dec-2013）
/dev/vda5 was not cleanly unmounted, check forced.
Pass 1: Checking inodes, blocks, and sizes
Deleted inode 1577 has zero dtime.  Fix&lt;y&gt;? yes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information

/dev/vda5: ***** FILE SYSTEM WAS MODIFIED *****  # 文件系统被改过，所以这里会有警告！
/dev/vda5: 11/65536 files （0.0% non-contiguous）, 12955/262144 blocks
# 好巧合！鸟哥使用这个方式来检验系统，恰好遇到文件系统出问题！于是可以有比较多的解释方向！
# 当文件系统出问题，它就会要你选择是否修复～如果修复如上所示，按下 y 即可！
# 最终系统会告诉你，文件系统已经被更改过，要注意该项目的意思！

范例：已默认设置强制检查一次 /dev/vda5
[root@study ~]# fsck.ext4 /dev/vda5
e2fsck 1.42.9 （28-Dec-2013）
/dev/vda5: clean, 11/65536 files, 12955/262144 blocks
# 文件系统状态正常，它并不会进入强制检查！会告诉你文件系统没问题 （clean）

[root@study ~]# fsck.ext4 -f /dev/vda5
e2fsck 1.42.9 （28-Dec-2013）
Pass 1: Checking inodes, blocks, and sizes
....（下面省略）....
```

无论是 xfs_repair 或 fsck.ext4，这都是用来检查与修正文件系统错误的指令。注意：通常只有身为 root 且你的文件系统有问题的时候才使用这个指令，否则在正常状况下使用此一指令， 可能会造成对系统的危害！通常使用这个指令的场合都是在系统出现极大的问题，导致你在 Linux 开机的时候得进入单人单机模式下进行维护的行为时，才必须使用此一指令！

另外，如果你怀疑刚刚格式化成功的磁盘有问题的时后，也可以使用 xfs_repair/fsck.ext4 来检查一磁盘呦！其实就有点像是 Windows 的 scandisk 啦！此外，由于 xfs_repair/fsck.ext4 在扫瞄磁盘的时候，可能会造成部分 filesystem 的修订，所以“执行 xfs_repair/fsck.ext4 时，被检查的 partition 务必不可挂载到系统上！亦即是需要在卸载的状态喔！”

7.3.5 文件系统挂载与卸载

我们在本章一开始时的挂载点的意义当中提过挂载点是目录， 而这个目录是进入磁盘分区（其实是文件系统啦！）的入口就是了。不过要进行挂载前，你最好先确定几件事：

- 单一文件系统不应该被重复挂载在不同的挂载点（目录）中；

- 单一目录不应该重复挂载多个文件系统；
- 要作为挂载点的目录，理论上应该都是空目录才是。

尤其是上述的后两点！如果你要用来挂载的目录里面并不是空的，那么挂载了文件系统之后，原目录下的东西就会暂时的消失。 举个例子来说，假设你的 /home 原本与根目录 (/) 在同一个文件系统中，下面原本就有 /home/test 与 /home/vbird 两个目录。然后你想要加入新的磁盘，并且直接挂载 /home 下面，那么当你挂载上新的分区时，则 /home 目录显示的是新分区内的数据，至于原先的 test 与 vbird 这两个目录就会暂时的被隐藏掉了！注意喔！并不是被覆盖掉， 而是暂时的隐藏了起来，等到新分区被卸载之后，则 /home 原本的内容就会再次的跑出来啦！

而要将文件系统挂载到我们的 Linux 系统上，就要使用 mount 这个指令啦！ 不过，这个指令真的是博大精深～粉难啦！我们学简单一点啊～ ^_^

```
[root@study ~]# mount -a
[root@study ~]# mount [-l]
[root@study ~]# mount [-t 文件系统] LABEL=''  挂载点
[root@study ~]# mount [-t 文件系统] UUID=''   挂载点  # 鸟哥近期建议用这种方式喔！
[root@study ~]# mount [-t 文件系统] 设备文件名  挂载点
选项与参数：
-a  ：依照配置文件 [/etc/fstab](../Text/index.html#fstab) 的数据将所有未挂载的磁盘都挂载上来
-l  ：单纯的输入 mount 会显示目前挂载的信息。加上 -l 可增列 Label 名称！
-t  ：可以加上文件系统种类来指定欲挂载的类型。常见的 Linux 支持类型有：xfs, ext3, ext4,
      reiserfs, vfat, iso9660(光盘格式), nfs, cifs, smbfs (后三种为网络文件系统类型)
-n  ：在默认的情况下，系统会将实际挂载的情况实时写入 /etc/mtab 中，以利其他程序的运行。
      但在某些情况下(例如单人维护模式)为了避免问题会刻意不写入。此时就得要使用 -n 选项。
-o  ：后面可以接一些挂载时额外加上的参数！比方说帐号、密码、读写权限等：
      async, sync:   此文件系统是否使用同步写入 (sync) 或非同步 (async) 的
                     内存机制，请参考[文件系统运行方式](../Text/index.html#harddisk-filerun)。
      atime,noatime: 是否修订文件的读取时间(atime)。为了性能，某些时刻可使用 noatime
      ro, rw:        挂载文件系统成为只读(ro) 或可读写(rw)
      auto, noauto:  允许此 filesystem 被以 mount -a 自动挂载(auto)
      dev, nodev:    是否允许此 filesystem 上，可创建设备文件？ dev 为可允许
      suid, nosuid:  是否允许此 filesystem 含有 suid/sgid 的文件格式？
      exec, noexec:  是否允许此 filesystem 上拥有可执行 binary 文件？
      user, nouser:  是否允许此 filesystem 让任何使用者执行 mount ？一般来说，
                     mount 仅有 root 可以进行，但下达 user 参数，则可让
                     一般 user 也能够对此 partition 进行 mount 。
      defaults:      默认值为：rw, suid, dev, exec, auto, nouser, and async
      remount:       重新挂载，这在系统出错，或重新更新参数时，很有用！
```

基本上，CentOS 7 已经太聪明了，因此你不需要加上 -t 这个选项，系统会自动的分析最恰当的文件系统来尝试挂载你需要的设备！ 这也是使用 blkid 就能够显示正确的文件系统的缘故！ 那 CentOS 是怎么找出文件系统类型的呢？ 由于文件系统几乎都有 superblock ，我们的 Linux 可以通过分析 superblock 搭配 Linux 自己的驱动程序去测试挂载， 如果成功的套和了，就立刻自动的使用该类型的文件系统挂载起来啊！那么系统有没有指定哪些类型的 filesystem 才需要进行上述的挂载测试呢？ 主要是参考下面这两个文件：

- /etc/filesystems：系统指定的测试挂载文件系统类型的优先顺序；
- /proc/filesystems：Linux系统已经载入的文件系统类型。

那我怎么知道我的 Linux 有没有相关文件系统类型的驱动程序呢？我们 Linux 支持的文件系统之驱动程序都写在如下的目录中：

- /lib/modules/$（uname -r）/kernel/fs/

例如 ext4 的驱动程序就写在“/lib/modules/$（uname -r）/kernel/fs/ext4/”这个目录下啦！

另外，过去我们都习惯使用设备文件名然后直接用该文件名挂载， 不过近期以来鸟哥比较建议使用 UUID 来识别文件系统，会比设备名称与标头名称还要更可靠！因为是独一无二的啊！

- 挂载 xfs/ext4/vfat 等文件系统

```
范例：找出 /dev/vda4 的 UUID 后，用该 UUID 来挂载文件系统到 /data/xfs 内
[root@study ~]# blkid /dev/vda4
/dev/vda4: UUID="e0a6af55-26e7-4cb7-a515-826a8bd29e90" TYPE="xfs"

[root@study ~]# mount UUID="e0a6af55-26e7-4cb7-a515-826a8bd29e90" /data/xfs
mount: mount point /data/xfs does not exist  # 非正规目录！所以手动创建它！

[root@study ~]# mkdir -p /data/xfs
[root@study ~]# mount UUID="e0a6af55-26e7-4cb7-a515-826a8bd29e90" /data/xfs
[root@study ~]# df /data/xfs
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/vda4        1038336 32864   1005472   4% /data/xfs
# 顺利挂载，且容量约为 1G 左右没问题！

范例：使用相同的方式，将 /dev/vda5 挂载于 /data/ext4
[root@study ~]# blkid /dev/vda5
/dev/vda5: UUID="899b755b-1da4-4d1d-9b1c-f762adb798e1" TYPE="ext4"

[root@study ~]# mkdir /data/ext4
[root@study ~]# mount UUID="899b755b-1da4-4d1d-9b1c-f762adb798e1" /data/ext4
[root@study ~]# df /data/ext4
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/vda5         999320  2564    927944   1% /data/ext4
```

- 挂载 CD 或 DVD 光盘

请拿出你的 CentOS 7 原版光盘出来，然后放入到光驱当中，我们来测试一下这个玩意儿啰！

```
范例：将你用来安装 Linux 的 CentOS 原版光盘拿出来挂载到 /data/cdrom！
[root@study ~]# blkid
.....（前面省略）.....
/dev/sr0: UUID="2015-04-01-00-21-36-00" LABEL="CentOS 7 x86_64" TYPE="iso9660" PTTYPE="do

[root@study ~]# mkdir /data/cdrom
[root@study ~]# mount /dev/sr0 /data/cdrom
mount: /dev/sr0 is write-protected, mounting read-only

[root@study ~]# df /data/cdrom
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sr0         7413478 7413478         0 100% /data/cdrom
# 怎么会使用掉 100% 呢？是啊！因为是 DVD 啊！所以无法再写入了啊！
```

光驱一挂载之后就无法退出光盘片了！除非你将他卸载才能够退出！ 从上面的数据你也可以发现，因为是光盘嘛！所以磁盘使用率达到 100% ，因为你无法直接写入任何数据到光盘当中！ 此外，如果你使用的是图形界面，那么系统会自动的帮你挂载这个光盘到 /media/ 里面去喔！也可以不卸载就直接退出！ 但是文字界面没有这个福利就是了！ ^_^



Tips 话说当时年纪小 （其实是刚接触 Linux 的那一年, 1999 年前后），摸 Linux 到处碰壁！连将 CDROM 挂载后， 光驱竟然都不让我退片！那个时候难过的要死！还用回纹针插入光驱让光盘退片耶！不过如此一来光盘就无法被使用了！ 若要再次使用光驱，当时的解决的方法竟然是“重新开机！”囧的可以啊！

- 挂载 vfat 中文U盘 （USB磁盘）

请拿出你的U盘并插入 Linux 主机的 USB接口中！注意，你的这个U盘不能够是 NTFS 的文件系统喔！接下来让我们测试测试吧！

```
范例：找出你的U盘设备的 UUID，并挂载到 /data/usb 目录中
[root@study ~]# blkid
/dev/sda1: UUID="35BC-6D6B" TYPE="vfat"

[root@study ~]# mkdir /data/usb
[root@study ~]#   mount -o codepage=950,iocharset=utf8 UUID="35BC-6D6B" /data/usb
[root@study ~]# # mount -o codepage=950,iocharset=big5 UUID="35BC-6D6B" /data/usb
[root@study ~]# df /data/usb
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/sda1        2092344     4   2092340   1% /data/usb
```

如果带有中文文件名的数据，那么可以在挂载时指定一下挂载文件系统所使用的语系数据。 在 man mount 找到 vfat 文件格式当中可以使用 codepage 来处理！中文语系的代码为 950 喔！另外，如果想要指定中文是万国码还是大五码， 就得要使用 iocharset 为 utf8 还是 big5 两者择一了！因为鸟哥的U盘使用 utf8 编码，因此将上述的 big5 前面加上 # 符号， 代表注解该行的意思啰！

万一你使用的 USB 磁盘被格式化为 NTFS 时，那可能就得要动点手脚，因为默认的 CentOS 7 并没有支持 NTFS 文件系统格式！ 所以你得要安装 NTFS 文件系统的驱动程序后，才有办法处理的！这部份我们留待 22 章讲到 yum 服务器时再来谈吧！ 因为目前我们也还没有网络、也没有讲软件安装啊！ ^_^

- 重新挂载根目录与挂载不特定目录

整个目录树最重要的地方就是根目录了，所以根目录根本就不能够被卸载的！问题是，如果你的挂载参数要改变， 或者是根目录出现“只读”状态时，如何重新挂载呢？最可能的处理方式就是重新开机 （reboot）！ 不过你也可以这样做：

```
范例：将 / 重新挂载，并加入参数为 rw 与 auto
[root@study ~]# mount -o remount,rw,auto /
```

重点是那个“ -o remount,xx ”的选项与参数！请注意，要重新挂载 （remount） 时， 这是个非常重要的机制！尤其是当你进入单人维护模式时，你的根目录常会被系统挂载为只读，这个时候这个指令就太重要了！

另外，我们也可以利用 mount 来将某个目录挂载到另外一个目录去喔！这并不是挂载文件系统，而是额外挂载某个目录的方法！ 虽然下面的方法也可以使用 symbolic link 来链接，不过在某些不支持符号链接的程序运行中，还是得要通过这样的方法才行。

```
范例：将 /var 这个目录暂时挂载到 /data/var 下面：
[root@study ~]# mkdir /data/var
[root@study ~]# mount --bind /var /data/var
[root@study ~]# ls -lid /var /data/var
16777346 drwxr-xr-x. 22 root root 4096 Jun 15 23:43 /data/var
16777346 drwxr-xr-x. 22 root root 4096 Jun 15 23:43 /var
# 内容完全一模一样啊！因为挂载目录的缘故！

[root@study ~]# mount &#124; grep var
/dev/mapper/centos-root on /data/var type xfs （rw,relatime,seclabel,attr2,inode64,noquota
```

看起来，其实两者链接到同一个 inode 嘛！ ^_^ 没错啦！通过这个 mount --bind 的功能， 您可以将某个目录挂载到其他目录去喔！而并不是整块 filesystem 的啦！所以从此进入 /data/var 就是进入 /var 的意思喔！

- umount （将设备文件卸载）

```
[root@study ~]# umount [-fn] 设备文件名或挂载点
选项与参数：
-f  ：强制卸载！可用在类似网络文件系统 （NFS） 无法读取到的情况下；
-l  ：立刻卸载文件系统，比 -f 还强！
-n  ：不更新 /etc/mtab 情况下卸载。
```

就是直接将已挂载的文件系统给他卸载即是！卸载之后，可以使用 df 或 mount 看看是否还存在目录树中？ 卸载的方式，可以下达设备文件名或挂载点，均可接受啦！下面的范例做看看吧！

```
范例：将本章之前自行挂载的文件系统全部卸载：
[root@study ~]# mount
.....（前面省略）.....
/dev/vda4 on /data/xfs type xfs （rw,relatime,seclabel,attr2,inode64,logbsize=256k,sunit=
/dev/vda5 on /data/ext4 type ext4 （rw,relatime,seclabel,data=ordered）
/dev/sr0 on /data/cdrom type iso9660 （ro,relatime）
/dev/sda1 on /data/usb type vfat （rw,relatime,fmask=0022,dmask=0022,codepage=950,iocharse
/dev/mapper/centos-root on /data/var type xfs （rw,relatime,seclabel,attr2,inode64,noquota
# 先找一下已经挂载的文件系统，如上所示，特殊字体即为刚刚挂载的设备啰！
# 基本上，卸载后面接设备或挂载点都可以！不过最后一个 centos-root 由于有其他挂载，
# 因此，该项目一定要使用挂载点来卸载才行！

[root@study ~]# umount /dev/vda4      &lt;==用设备文件名来卸载
[root@study ~]# umount /data/ext4     &lt;==用挂载点来卸载
[root@study ~]# umount /data/cdrom    &lt;==因为挂载点比较好记忆！
[root@study ~]# umount /data/usb
[root@study ~]# umount /data/var      &lt;==一定要用挂载点！因为设备有被其他方式挂载
```

由于通通卸载了，此时你才可以退出光盘片、软盘片、U盘等设备喔！如果你遇到这样的情况：

```
[root@study ~]# mount /dev/sr0 /data/cdrom
[root@study ~]# cd /data/cdrom
[root@study cdrom]# umount /data/cdrom
umount: /data/cdrom: target is busy.
        （In some cases useful info about processes that use
         the device is found by lsof（8） or fuser（1））

[root@study cdrom]# cd /
[root@study /]# umount /data/cdrom
```

由于你目前正在 /data/cdrom/ 的目录内，也就是说其实“你正在使用该文件系统”的意思！所以自然无法卸载这个设备！那该如何是好？就“离开该文件系统的挂载点”即可。以上述的案例来说， 你可以使用“ cd / ”回到根目录，就能够卸载 /data/cdrom 啰！简单吧！

7.3.6 磁盘/文件系统参数修订

某些时刻，你可能会希望修改一下目前文件系统的一些相关信息，举例来说，你可能要修改 Label name ， 或者是 journal 的参数，或者是其他磁盘/文件系统运行时的相关参数 （例如 DMA 启动与否～）。 这个时候，就得需要下面这些相关的指令功能啰～

- mknod

还记得我们说过，在 Linux 下面所有的设备都以文件来代表吧！但是那个文件如何代表该设备呢？ 很简单！就是通过文件的 major 与 minor 数值来替代的～所以，那个 major 与 minor 数值是有特殊意义的，不是随意设置的喔！我们在 lsblk 指令的用法里面也谈过这两个数值呢！举例来说，在鸟哥的这个测试机当中， 那个用到的磁盘 /dev/vda 的相关设备代码如下：

```
[root@study ~]# ll /dev/vda*
brw-rw----. 1 root disk 252, 0 Jun 24 02:30 /dev/vda
brw-rw----. 1 root disk 252, 1 Jun 24 02:30 /dev/vda1
brw-rw----. 1 root disk 252, 2 Jun 15 23:43 /dev/vda2
brw-rw----. 1 root disk 252, 3 Jun 15 23:43 /dev/vda3
brw-rw----. 1 root disk 252, 4 Jun 24 20:00 /dev/vda4
brw-rw----. 1 root disk 252, 5 Jun 24 21:15 /dev/vda5
```

上表当中 252 为主要设备代码 （Major） 而 0~5 则为次要设备代码 （Minor）。 我们的 Linux 核心认识的设备数据就是通过这两个数值来决定的！举例来说，常见的磁盘文件名 /dev/sda 与 /dev/loop0 设备代码如下所示：

磁盘文件名	Major	Minor
/dev/sda	8	0-15
/dev/sdb	8	16-31
/dev/loop0	7	0
/dev/loop1	7	1

如果你想要知道更多核心支持的硬件设备代码 （major, minor） 请参考核心官网的链接[8]。 基本上，Linux 核心 2.6 版以后，硬件文件名已经都可以被系统自动的实时产生了，我们根本不需要手动创建设备文件。 不过某些情况下面我们可能还是得要手动处理设备文件的，例如在某些服务被关到特定目录下时（chroot）， 就需要这样做了。此时这个 mknod 就得要知道如何操作才行！

```
[root@study ~]# mknod 设备文件名 [bcp] [Major] [Minor]
选项与参数：
设备种类：
   b  ：设置设备名称成为一个周边储存设备文件，例如磁盘等；
   c  ：设置设备名称成为一个周边输入设备文件，例如鼠标/键盘等；
   p  ：设置设备名称成为一个 FIFO 文件；
Major ：主要设备代码；
Minor ：次要设备代码；

范例：由上述的介绍我们知道 /dev/vda10 设备代码 252, 10，请创建并查阅此设备
[root@study ~]# mknod /dev/vda10 b 252 10
[root@study ~]# ll /dev/vda10
brw-r--r--. 1 root root 252, 10 Jun 24 23:40 /dev/vda10
# 上面那个 252 与 10 是有意义的，不要随意设置啊！

范例：创建一个 FIFO 文件，文件名为 /tmp/testpipe
[root@study ~]# mknod /tmp/testpipe p
[root@study ~]# ll /tmp/testpipe
prw-r--r--. 1 root root 0 Jun 24 23:44 /tmp/testpipe
# 注意啊！这个文件可不是一般文件，不可以随便就放在这里！
# 测试完毕之后请删除这个文件吧！看一下这个文件的类型！是 p 喔！^_^

[root@study ~]# rm /dev/vda10 /tmp/testpipe
rm: remove block special file '/dev/vda10' ? y
rm: remove fifo '/tmp/testpipe' ? y
```

- xfs_admin 修改 XFS 文件系统的 UUID 与 Label name

如果你当初格式化的时候忘记加上标头名称，后来想要再次加入时，不需要重复格式化！直接使用这个 xfs_admin 即可。 这个指令直接拿来处理 LABEL name 以及 UUID 即可啰！

```
[root@study ~]# xfs_admin [-lu] [-L label] [-U uuid] 设备文件名
选项与参数：
-l  ：列出这个设备的 label name
-u  ：列出这个设备的 UUID
-L  ：设置这个设备的 Label name
-U  ：设置这个设备的 UUID 喔！

范例：设置 /dev/vda4 的 label name 为 vbird_xfs，并测试挂载
[root@study ~]# xfs_admin -L vbird_xfs /dev/vda4
writing all SBs
new label = "vbird_xfs"                 # 产生新的 LABEL 名称啰！
[root@study ~]# xfs_admin -l /dev/vda4
label = "vbird_xfs"
[root@study ~]# mount LABEL=vbird_xfs /data/xfs/

范例：利用 uuidgen 产生新 UUID 来设置 /dev/vda4，并测试挂载
[root@study ~]# umount /dev/vda4       # 使用前，请先卸载！
[root@study ~]# uuidgen
e0fa7252-b374-4a06-987a-3cb14f415488    # 很有趣的指令！可以产生新的 UUID 喔！
[root@study ~]# xfs_admin -u /dev/vda4
UUID = e0a6af55-26e7-4cb7-a515-826a8bd29e90
[root@study ~]# xfs_admin -U e0fa7252-b374-4a06-987a-3cb14f415488 /dev/vda4
Clearing log and setting UUID
writing all SBs
new UUID = e0fa7252-b374-4a06-987a-3cb14f415488
[root@study ~]# mount UUID=e0fa7252-b374-4a06-987a-3cb14f415488 /data/xfs
```

不知道你会不会有这样的疑问：“鸟哥啊，既然 mount 后面使用设备文件名 （/dev/vda4） 也可以挂载成功，那你为什么要用很讨厌的很长一串的 UUID 来作为你的挂载时写入的设备名称啊？”问的好！原因是这样的：“因为你没有办法指定这个磁盘在所有的 Linux 系统中，文件名一定都会是 /dev/vda ！”

举例来说，我们刚刚使用的U盘在鸟哥这个测试系统当中查询到的文件名是 /dev/sda，但是当这个U盘放到其他的已经有 /dev/sda 文件名的 Linux 系统下，它的文件名就会被指定成为 /dev/sdb 或 /dev/sdc 等等。反正，不会是 /dev/sda 了！那我怎么用同一个指令去挂载这只U盘呢？ 当然有问题吧！但是 UUID 可是很难重复的！看看上面 uuidgen 产生的结果你就知道了！所以你可以确定该名称不会被重复！ 这对系统管理上可是相当有帮助的！它也比 LABEL name 要更精准的多呢！ ^_^

- tune2fs 修改 ext4 的 label name 与 UUID

```
[root@study ~]# tune2fs [-l] [-L Label] [-U uuid] 设备文件名
选项与参数：
-l  ：类似 dumpe2fs -h 的功能～将 superblock 内的数据读出来～
-L  ：修改 LABEL name
-U  ：修改 UUID 啰！

范例：列出 /dev/vda5 的 label name 之后，将它改成 vbird_ext4
[root@study ~]# dumpe2fs -h /dev/vda5 &#124; grep name
dumpe2fs 1.42.9 (28-Dec-2013)
Filesystem volume name:   &lt;none&gt;   # 果然是没有设置的！

[root@study ~]# tune2fs -L vbird_ext4 /dev/vda5
[root@study ~]# dumpe2fs -h /dev/vda5 &#124; grep name
Filesystem volume name:   vbird_ext4
[root@study ~]# mount LABEL=vbird_ext4 /data/ext4
```

这个指令的功能其实很广泛啦～上面鸟哥仅列出很简单的一些参数而已，更多的用法请自行参考 man tune2fs 。

7.4 设置开机挂载

手动处理 mount 不是很人性化，我们总是需要让系统“自动”在开机时进行挂载的！本小节就是在谈这玩意儿！ 另外，从 FTP 服务器捉下来的镜像文件能否不用烧录就可以读取内容？我们也需要谈谈先！

7.4.1 开机挂载 /etc/fstab 及 /etc/mtab

刚刚上面说了许多，那么可不可以在开机的时候就将我要的文件系统都挂好呢？这样我就不需要每次进入 Linux 系统都还要在挂载一次呀！当然可以啰！那就直接到 /etc/fstab 里面去修修就行啰！不过，在开始说明前，这里要先跟大家说一说系统挂载的一些限制：

- 根目录 / 是必须挂载的﹐而且一定要先于其它 mount point 被挂载进来。
- 其它 mount point 必须为已创建的目录﹐可任意指定﹐但一定要遵守必须的系统目录架构原则 （FHS）
- 所有 mount point 在同一时间之内﹐只能挂载一次。
- 所有 partition 在同一时间之内﹐只能挂载一次。
- 如若进行卸载﹐您必须先将工作目录移到 mount point（及其子目录） 之外。

让我们直接查阅一下 /etc/fstab 这个文件的内容吧！

```
[root@study ~]# cat /etc/fstab
# Device                              Mount point  filesystem parameters    dump fsck
/dev/mapper/centos-root                   /       xfs     defaults           0 0
UUID=94ac5f77-cb8a-495e-a65b-2ef7442b837c /boot   xfs     defaults           0 0
/dev/mapper/centos-home                   /home   xfs     defaults           0 0
/dev/mapper/centos-swap                   swap    swap    defaults           0 0
```

其实 /etc/fstab （filesystem table） 就是将我们利用 mount 指令进行挂载时， 将所有的选项与参数写入到这个文件中就是了。除此之外， /etc/fstab 还加入了 dump 这个备份用指令的支持！ 与开机时是否进行文件系统检验 fsck 等指令有关。 这个文件的内容共有六个字段，这六个字段非常的重要！你“一定要背起来”才好！ 各个字段的总结数据与详细数据如下：



Tips 鸟哥比较龟毛一点，因为某些 distributions 的 /etc/fstab 文件排列方式蛮丑的， 虽然每一栏之间只要以空白字符分开即可，但就是觉得丑，所以通常鸟哥就会自己排列整齐， 并加上注解符号（就是 # ），来帮我记忆这些信息！

```
[设备/UUID等]  [挂载点]  [文件系统]  [文件系统参数]  [dump]  [fsck]
```

- 第一栏：磁盘设备文件名/UUID/LABEL name：

这个字段可以填写的数据主要有三个项目：

- 文件系统或磁盘的设备文件名，如 /dev/vda2 等
- 文件系统的 UUID 名称，如 UUID=xxx
- 文件系统的 LABEL 名称，例如 LABEL=xxx

因为每个文件系统都可以有上面三个项目，所以你喜欢哪个项目就填哪个项目！无所谓的！只是从鸟哥测试机的 /etc/fstab 里面看到的，在挂载点 /boot 使用的已经是 UUID 了喔！那你会说不是还有多个写 /dev/mapper/xxx 的吗？怎么回事啊？ 因为那个是 LVM 啊！LVM 的文件名在你的系统中也算是独一无二的，这部份我们在后续章节再来谈。 不过，如果为了一致性，你还是可以将他改成 UUID 也没问题喔！（鸟哥还是比较建议使用 UUID 喔！） 要记得使用 blkid 或 xfs_admin 来查询 UUID 喔！

- 第二栏：挂载点 （mount point）：：

就是挂载点啊！挂载点是什么？一定是目录啊～要知道啊！忘记的话，请回本章稍早之前的数据瞧瞧喔！

- 第三栏：磁盘分区的文件系统：

在手动挂载时可以让系统自动测试挂载，但在这个文件当中我们必须要手动写入文件系统才行！ 包括 xfs, ext4, vfat, reiserfs, nfs 等等。

- 第四栏：文件系统参数：

记不记得我们在 mount 这个指令中谈到很多特殊的文件系统参数？ 还有我们使用过的“-o codepage=950”？这些特殊的参数就是写入在这个字段啦！ 虽然之前在 mount 已经提过一次，这里我们利用表格的方式再汇整一下：

参数	内容意义
async/sync 非同步/同步	设置磁盘是否以非同步方式运行！默认为 async（性能较佳）
auto/noauto 自动/非自动	当下达 mount -a 时，此文件系统是否会被主动测试挂载。默认为 auto。
rw/ro 可读写/只读	让该分区以可读写或者是只读的型态挂载上来，如果你想要分享的数据是不给使用者随意变更的， 这里也能够设置为只读。则不论在此文件系统的文件是否设置 w 权限，都无法写入喔！
exec/noexec 可执行/不可执行	限制在此文件系统内是否可以进行“执行”的工作？如果是纯粹用来储存数据的目录， 那么可以设置为 noexec 会比较安全。不过，这个参数也不能随便使用，因为你不知道该目录下是否默认会有可执行文件。举例来说，如果你将 noexec 设置在 /var ，当某些软件将一些可执行文件放置于 /var 下时，那就会产生很大的问题喔！ 因此，建议这个 noexec 最多仅设置于你自订或分享的一般数据目录。
user/nouser 允许/不允许使用者挂载	是否允许使用者使用 mount指令来挂载呢？一般而言，我们当然不希望一般身份的 user 能使用 mount 啰，因为太不安全了，因此这里应该要设置为 nouser 啰！
suid/nosuid 具有/不具有 suid 权限	该文件系统是否允许 SUID 的存在？如果不是可执行文件放置目录，也可以设置为 nosuid 来取消这个功能！
defaults	同时具有 **rw, suid, dev, exec, auto, nouser, async** 等参数。 基本上，默认情况使用 defaults 设置即可！

- 第五栏：能否被 dump 备份指令作用：

dump 是一个用来做为备份的指令，不过现在有太多的备份方案了，所以这个项目可以不要理会啦！直接输入 0 就好了！

- 第六栏：是否以 fsck 检验扇区：

早期开机的流程中，会有一段时间去检验本机的文件系统，看看文件系统是否完整（clean）。 不过这个方式使用的主要是通过 fsck 去做的，我们现在用的 xfs 文件系统就没有办法适用，因为 xfs 会自己进行检验，不需要额外进行这个动作！所以直接填 0 就好了。

好了，那么让我们来处理一下我们的新建的文件系统，看看能不能开机就挂载呢？

例题：假设我们要将 /dev/vda4 每次开机都自动挂载到 /data/xfs ，该如何进行？答：首先，请用 nano 将下面这一行写入 /etc/fstab 最后面中；

```
[root@study ~]# nano /etc/fstab
UUID="e0fa7252-b374-4a06-987a-3cb14f415488"  /data/xfs  xfs  defaults  0 0
```

再来看看 /dev/vda4 是否已经挂载，如果挂载了，请务必卸载再说！

```
[root@study ~]# df
Filesystem              1K-blocks    Used Available Use% Mounted on
/dev/vda4                 1038336   32864   1005472   4% /data/xfs
# 竟然不知道何时被挂载了？赶紧给他卸载先！
# **因为，如果要被挂载的文件系统已经被挂载了（无论挂载在哪个目录），那测试就不会进行喔！**

[root@study ~]# umount /dev/vda4
```

最后测试一下刚刚我们写入 /etc/fstab 的语法有没有错误！这点很重要！因为这个文件如果写错了， 则你的 Linux 很可能将无法顺利开机完成！所以请务必要测试测试喔！

```
[root@study ~]# mount -a
[root@study ~]# df /data/xfs
```

最终有看到 /dev/vda4 被挂载起来的信息才是成功的挂载了！而且以后每次开机都会顺利的将此文件系统挂载起来的！ 现在，你可以下达 reboot 重新开机，然后看一下默认有没有多一个 /dev/vda4 呢？

/etc/fstab 是开机时的配置文件，不过，实际 filesystem 的挂载是记录到 /etc/mtab 与 /proc/mounts 这两个文件当中的。每次我们在更动 filesystem 的挂载时，也会同时更动这两个文件喔！但是，万一发生你在 /etc/fstab 输入的数据错误，导致无法顺利开机成功，而进入单人维护模式当中，那时候的 / 可是 read only 的状态，当然你就无法修改 /etc/fstab ，也无法更新 /etc/mtab 啰～那怎么办？没关系，可以利用下面这一招：

```
[root@study ~]# mount -n -o remount,rw /
```

7.4.2 特殊设备 loop 挂载 （镜像文件不烧录就挂载使用）

如果有光盘镜像文件，或者是使用文件作为磁盘的方式时，那就得要使用特别的方法来将他挂载起来，不需要烧录啦！

- 挂载光盘/DVD镜像文件

想像一下如果今天我们从国家高速网络中心（http://ftp.twaren.net）或者是昆山科大（http://ftp.ksu.edu.tw）下载了 Linux 或者是其他所需光盘/DVD的镜像文件后， 难道一定需要烧录成为光盘才能够使用该文件里面的数据吗？当然不是啦！我们可以通过 loop 设备来挂载的！

那要如何挂载呢？鸟哥将整个 CentOS 7.x 的 DVD 镜像文件捉到测试机上面，然后利用这个文件来挂载给大家参考看看啰！

```
[root@study ~]# ll -h /tmp/CentOS-7.0-1406-x86_64-DVD.iso
-rw-r--r--. 1 root root 3.9G Jul  7  2014 /tmp/CentOS-7.0-1406-x86_64-DVD.iso
# 看到上面的结果吧！这个文件就是镜像文件，文件非常的大吧！

[root@study ~]# mkdir /data/centos_dvd
[root@study ~]# mount -o loop /tmp/CentOS-7.0-1406-x86_64-DVD.iso /data/centos_dvd
[root@study ~]# df /data/centos_dvd
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/loop0       4050860 4050860         0 100% /data/centos_dvd
# 就是这个项目！ .iso 镜像文件内的所有数据可以在 /data/centos_dvd 看到！

[root@study ~]# ll /data/centos_dvd
total 607
-rw-r--r--. 1  500  502     14 Jul  5  2014 CentOS_BuildTag &lt;==瞧！就是DVD的内容啊！
drwxr-xr-x. 3  500  502   2048 Jul  4  2014 EFI
-rw-r--r--. 1  500  502    611 Jul  5  2014 EULA
-rw-r--r--. 1  500  502  18009 Jul  5  2014 GPL
drwxr-xr-x. 3  500  502   2048 Jul  4  2014 images
.....（下面省略）.....

[root@study ~]# umount /data/centos_dvd/
# 测试完成！记得将数据给他卸载！同时这个镜像文件也被鸟哥删除了...测试机容量不够大！
```

非常方便吧！如此一来我们不需要将这个文件烧录成为光盘或者是 DVD 就能够读取内部的数据了！ 换句话说，你也可以在这个文件内“动手脚”去修改文件的！这也是为什么很多镜像文件提供后，还得要提供验证码 （MD5） 给使用者确认该镜像文件没有问题！

- 创建大文件以制作 loop 设备文件！

想一想，既然能够挂载 DVD 的镜像文件，那么我能不能制作出一个大文件，然后将这个文件格式化后挂载呢？ 好问题！这是个有趣的动作！而且还能够帮助我们解决很多系统的分区不良的情况呢！举例来说，如果当初在分区时， 你只有分区出一个根目录，假设你已经没有多余的容量可以进行额外的分区的！偏偏根目录的容量还很大！ 此时你就能够制作出一个大文件，然后将这个文件挂载！如此一来感觉上你就多了一个分区啰！用途非常的广泛啦！

下面我们在 /srv 下创建一个 512MB 左右的大文件，然后将这个大文件格式化并且实际挂载来玩一玩！ 这样你会比较清楚鸟哥在讲啥！

- 创建大型文件

首先，我们得先有一个大的文件吧！怎么创建这个大文件呢？在 Linux 下面我们有一支很好用的程序 dd ！他可以用来创建空的文件喔！详细的说明请先翻到下一章 压缩指令的运用 来查阅，这里鸟哥仅作一个简单的范例而已。 假设我要创建一个空的文件在 /srv/loopdev ，那可以这样做：

```
[root@study ~]# dd if=/dev/zero of=/srv/loopdev bs=1M count=512
512+0 records in   &lt;==读入 512 笔数据
512+0 records out  &lt;==输出 512 笔数据
536870912 Bytes (537 MB) copied, 12.3484 seconds, 43.5 MB/s
# 这个指令的简单意义如下：
# if    是 input file ，输入文件。那个 /dev/zero 是会一直输出 0 的设备！
# of    是 output file ，将一堆零写入到后面接的文件中。
# bs    是每个 block 大小，就像文件系统那样的 block 意义；
# count 则是总共几个 bs 的意思。所以 bs*count 就是这个文件的容量了！

[root@study ~]# ll -h /srv/loopdev
-rw-r--r--. 1 root root 512M Jun 25 19:46 /srv/loopdev
```

dd 就好像在叠砖块一样，将 512 块，每块 1MB 的砖块堆叠成为一个大文件 （/srv/loopdev）！ 最终就会出现一个 512MB 的文件！粉简单吧！

- 大型文件的格式化

默认 xfs 不能够格式化文件的，所以要格式化文件得要加入特别的参数才行喔！让我们来瞧瞧！

```
[root@study ~]# mkfs.xfs -f /srv/loopdev
[root@study ~]# blkid /srv/loopdev
/srv/loopdev: UUID="7dd97bd2-4446-48fd-9d23-a8b03ffdd5ee" TYPE="xfs"
```

其实很简单啦！所以鸟哥就不输出格式化的结果了！要注意 UUID 的数值，未来会用到！

- 挂载

那要如何挂载啊？利用 mount 的特殊参数，那个 -o loop 的参数来处理！

```
[root@study ~]# mount -o loop UUID="7dd97bd2-4446-48fd-9d23-a8b03ffdd5ee" /mnt
[root@study ~]# df /mnt
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/loop0        520876 26372    494504   6% /mnt
```

通过这个简单的方法，感觉上你就可以在原本的分区在不更动原有的环境下制作出你想要的分区就是了！ 这东西很好用的！尤其是想要玩 Linux 上面的“虚拟机”的话， 也就是以一部 Linux 主机再切割成为数个独立的主机系统时，类似 VMware 这类的软件， 在 Linux 上使用 xen 这个软件，他就可以配合这种 loop device 的文件类型来进行根目录的挂载，真的非常有用的喔！ ^_^

比较特别的是，CentOS 7.x 越来越聪明了，现在你不需要下达 -o loop 这个选项与参数，它同样可以被系统挂上来！ 连直接输入 blkid 都会列出这个文件内部的文件系统耶！相当有趣！ 不过，为了考虑向下兼容性，鸟哥还是建议你加上 loop 比较妥当喔！ 现在，请将这个文件系统永远的自动挂载起来吧！

```
[root@study ~]# nano /etc/fstab
/srv/loopdev  /data/file  xfs  defaults**,loop**   0 0
# 毕竟系统大多仅查询 block device 去找出 UUID 而已，因此使用文件创建的 filesystem，
# 最好还是使用原本的文件名来处理，应该比较不容易出现错误讯息的！

[root@study ~]# umount /mnt
[root@study ~]# mkdir /data/file
[root@study ~]# mount -a
[root@study ~]# df /data/file
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/loop0        520876 26372    494504   6% /data/file
```

7.5 内存交换空间（swap）之创建

以前的年代因为内存不足，因此那个可以暂时将内存的程序拿到硬盘中暂放的内存交换空间（swap）就显的非常的重要！否则，如果突然间某支程序用掉你大部分的内存，那你的系统恐怕有损毁的情况发生喔！所以，早期在安装 Linux 之前，大家常常会告诉你：安装时一定需要的两个 partition ，一个是根目录，另外一个就是 swap（内存交换空间）。关于内存交换空间的解释在第三章安装 Linux 内的磁盘分区时有约略提过，请你自行回头瞧瞧吧！

一般来说，如果硬件的配备资源足够的话，那么 swap 应该不会被我们的系统所使用到，swap 会被利用到的时刻通常就是实体内存不足的情况了。从第零章的计算机概论当中，我们知道 CPU 所读取的数据都来自于内存，那当内存不足的时候，为了让后续的程序可以顺利的运行，因此在内存中暂不使用的程序与数据就会被挪到 swap 中了。此时内存就会空出来给需要执行的程序载入。由于 swap 是用磁盘来暂时放置内存中的信息，所以用到 swap 时，你的主机磁盘灯就会开始闪个不停啊！

虽然目前（2015）主机的内存都很大，至少都有 4GB 以上啰！因此在个人使用上，你不要设置 swap 在你的 Linux 应该也没有什么太大的问题。不过服务器可就不这么想了～由于你不会知道何时会有大量来自网络的要求，因此最好还是能够预留一些 swap 来缓冲一下系统的内存用量！至少达到“备而不用”的地步啊！

现在想像一个情况，你已经将系统创建起来了，此时却才发现你没有创建 swap ～那该如何是好呢？通过本章上面谈到的方法，你可以使用如下的方式来创建你的 swap 啰！

- 设置一个 swap partition
- 创建一个虚拟内存的文件

不啰唆，就立刻来处理处理吧！

7.5.1 使用实体分区创建swap

创建 swap 分区的方式也是非常的简单的！通过下面几个步骤就搞定啰：

1. 分区：先使用 gdisk 在你的磁盘中分区出一个分区给系统作为 swap 。由于 Linux 的 gdisk 默认会将分区的 ID 设置为 Linux 的文件系统，所以你可能还得要设置一下 system ID 就是了。
2. 格式化：利用创建 swap 格式的“mkswap 设备文件名”就能够格式化该分区成为 swap 格式啰
3. 使用：最后将该 swap 设备启动，方法为：“swapon 设备文件名”。
4. 观察：最终通过 free 与 swapon -s 这个指令来观察一下内存的用量吧！

不啰唆，立刻来实作看看！既然我们还有多余的磁盘容量可以分区，那么让我们继续分区出 512MB 的磁盘分区吧！然后将这个磁盘分区做成 swap 吧！

- 1. 先进行分区的行为啰！

```
[root@study ~]# gdisk /dev/vda
Command （? for help）: n
Partition number （6-128, default 6）:
First sector （34-83886046, default = 69220352） or {+-}size{KMGTP}:
Last sector （69220352-83886046, default = 83886046） or {+-}size{KMGTP}: +512M
Current type is 'Linux filesystem'
Hex code or GUID （L to show codes, Enter = 8300）: 8200
Changed type of partition to 'Linux swap'

Command （? for help）: p
Number  Start （sector）    End （sector）  Size       Code  Name
   6        69220352        70268927   512.0 MiB   8200  Linux swap  # 重点就是产生这东西！

Command （? for help）: w

Do you want to proceed? （Y/N）: y

[root@study ~]# partprobe
[root@study ~]# lsblk
NAME            MAJ:MIN RM   SIZE RO TYPE MOUNTPOINT
vda             252:0    0    40G  0 disk
.....（中间省略）.....
`-vda6          252:6    0   512M  0 part   # 确定这里是存在的才行！
# 鸟哥有简化输出喔！结果可以看到我们多了一个 /dev/vda6 可以使用于 swap 喔！
```

- 2. 开始创建 swap 格式

```
[root@study ~]# mkswap /dev/vda6
Setting up swapspace version 1, size = 524284 KiB
no label, UUID=6b17e4ab-9bf9-43d6-88a0-73ab47855f9d
[root@study ~]# blkid /dev/vda6
/dev/vda6: UUID="6b17e4ab-9bf9-43d6-88a0-73ab47855f9d" TYPE="swap"
# 确定格式化成功！且使用 blkid 确实可以抓到这个设备了喔！
```

- 3. 开始观察与载入看看吧！

```
[root@study ~]# free
              total        used        free      shared  buff/cache   available
Mem:        1275140      227244      330124        7804      717772      875536  # 实体内
Swap:       1048572      101340      947232                                      # swap 
# 我有 1275140K 的实体内存，使用 227244K 剩余 330124K ，使用掉的内存有
# 717772K 用在缓冲/高速缓存的用途中。至于 swap 已经有 1048572K 啰！这样会看了吧？！

[root@study ~]# swapon /dev/vda6
[root@study ~]# free
              total        used        free      shared  buff/cache   available
Mem:        1275140      227940      329256        7804      717944      874752
Swap:       1572856      101260     1471596     &lt;==有看到增加了没？

[root@study ~]# swapon -s
Filename                 Type            Size    Used    Priority
/dev/dm-1                partition       1048572 101260  -1
/dev/vda6                partition       524284  0       -2
# 上面列出目前使用的 swap 设备有哪些的意思！

[root@study ~]# nano /etc/fstab
UUID="6b17e4ab-9bf9-43d6-88a0-73ab47855f9d"  swap  swap  defaults  0  0
# 当然要写入配置文件，只不过不是文件系统，所以没有挂载点！第二个字段写入 swap 即可。
```

7.5.2 使用文件创建swap

如果是在实体分区无法支持的环境下，此时前一小节提到的 loop 设备创建方法就派的上用场啦！与实体分区不一样的，这个方法只是利用 dd 去创建一个大文件而已。多说无益，我们就再通过文件创建的方法创建一个 128 MB 的内存交换空间吧！

- 1. 使用 dd 这个指令来新增一个 128MB 的文件在 /tmp 下面：

```
[root@study ~]# dd if=/dev/zero of=/tmp/swap bs=1M count=128
128+0 records in
128+0 records out
134217728 Bytes （134 MB） copied, 1.7066 seconds, 78.6 MB/s

[root@study ~]# ll -h /tmp/swap
-rw-r--r--. 1 root root 128M Jun 26 17:47 /tmp/swap
```

这样一个 128MB 的文件就创建妥当。若忘记上述的各项参数的意义，请回前一小节查阅一下啰！

- 2. 使用 mkswap 将 /tmp/swap 这个文件格式化为 swap 的文件格式：

```
[root@study ~]# mkswap /tmp/swap
Setting up swapspace version 1, size = 131068 KiB
no label, UUID=4746c8ce-3f73-4f83-b883-33b12fa7337c
# 这个指令下达时请“特别小心”，因为下错字符控制，将可能使您的文件系统挂掉！
```

- 3. 使用 swapon 来将 /tmp/swap 启动啰！

```
[root@study ~]# swapon /tmp/swap
[root@study ~]# swapon -s
Filename            Type            Size    Used    Priority
/dev/dm-1           partition       1048572 100380  -1
/dev/vda6           partition       524284  0       -2
/tmp/swap           file            131068  0       -3
```

- 4. 使用 swapoff 关掉 swap file，并设置自动启用

```
[root@study ~]# nano /etc/fstab
/tmp/swap  swap  swap  defaults  0  0
# 为何这里不要使用 UUID 呢？这是因为系统仅会查询区块设备 （block device） 不会查询文件！
# 所以，这里千万不要使用 UUID，不然系统会查不到喔！

[root@study ~]# swapoff /tmp/swap /dev/vda6
[root@study ~]# swapon -s
Filename                                Type            Size    Used    Priority
/dev/dm-1                               partition       1048572 100380  -1
# 确定已经回复到原本的状态了！然后准备来测试！！

[root@study ~]# swapon -a
[root@study ~]# swapon -s
# 最终你又会看正确的三个 swap 出现啰！这也才确定你的 /etc/fstab 设置无误！
```

说实话，swap 在目前的桌面电脑来讲，存在的意义已经不大了！这是因为目前的 x86 主机所含的内存实在都太大了 （一般入门级至少也都有 4GB 了），所以，我们的 Linux 系统大概都用不到 swap 这个玩意儿的。不过， 如果是针对服务器或者是工作站这些常年上线的系统来说的话，那么，无论如何，swap 还是需要创建的。

因为 swap 主要的功能是当实体内存不够时，则某些在内存当中所占的程序会暂时被移动到 swap 当中，让实体内存可以被需要的程序来使用。另外，如果你的主机支持电源管理模式，也就是说，你的 Linux 主机系统可以进入“休眠”模式的话，那么， 运行当中的程序状态则会被纪录到 swap 去，以作为“唤醒”主机的状态依据！ 另外，有某些程序在运行时，本来就会利用 swap 的特性来存放一些数据段， 所以， swap 来是需要创建的！只是不需要太大！

7.6 文件系统的特殊观察与操作

文件系统实在是非常有趣的东西，鸟哥学了好几年还是很多东西不很懂呢！在学习的过程中很多朋友在讨论区都有提供一些想法！ 这些想法将他归纳起来有下面几点可以参考的数据呢！

7.6.1 磁盘空间之浪费问题

我们在前面的 EXT2 data block 介绍中谈到了一个 block 只能放置一个文件， 因此太多小文件将会浪费非常多的磁盘容量。但你有没有注意到，整个文件系统中包括 superblock, inode table 与其他中介数据等其实都会浪费磁盘容量喔！所以当我们在 /dev/vda4, /dev/vda5 创建起 xfs/ext4 文件系统时， 一挂载就立刻有很多容量被用掉了！

另外，不知道你有没有发现到，当你使用 ls -l 去查询某个目录下的数据时，第一行都会出现一个“total”的字样！ 那是啥东西？其实那就是该目录下的所有数据所耗用的实际 block 数量 * block 大小的值。 我们可以通过 ll -s 来观察看看上述的意义：

```
[root@study ~]# ll -sh
total 12K
4.0K -rw-------. 1 root root 1.8K May  4 17:57 anaconda-ks.cfg
4.0K -rw-r--r--. 2 root root  451 Jun 10  2014 crontab
 0 lrwxrwxrwx. 1 root root   12 Jun 23 22:31 crontab2 -&gt; /etc/crontab
4.0K -rw-r--r--. 1 root root 1.9K May  4 18:01 initial-setup-ks.cfg
 0 -rw-r--r--. 1 root root    0 Jun 16 01:11 test1
 0 drwxr-xr-x. 2 root root    6 Jun 16 01:11 test2
 0 -rw-rw-r--. 1 root root    0 Jun 16 01:12 test3
 0 drwxrwxr-x. 2 root root    6 Jun 16 01:12 test4
```

从上面的特殊字体部分，那就是每个文件所使用掉 block 的容量！举例来说，那个 crontab 虽然仅有 451Bytes ， 不过他却占用了整个 block （每个 block 为 4K），所以将所有的文件的所有的 block 加总就得到 12KBytes 那个数值了。 如果计算每个文件实际容量的加总结果，其实只有不到 5K 而已～所以啰，这样就耗费掉好多容量了！未来大家在讨论小磁盘、 大磁盘，文件大小的损耗时，要回想到这个区块喔！ ^_^

7.6.2 利用 GNU 的 parted 进行分区行为（Optional）

虽然你可以使用 gdisk/fdisk 很快速的将你的分区切割妥当，不过 gdisk 主要针对 GPT 而 fdisk 主要支持 MBR ，对 GPT 的支持还不够！ 所以使用不同的分区时，得要先查询到正确的分区表才能用适合的指令，好麻烦！有没有同时支持的指令呢？有的！那就是 parted 啰！



Tips 老实说，若不是后来有推出支持 GPT 的 gdisk，鸟哥其实已经爱用 parted 来进行分区行为了！虽然很多指令都需要同时开一个终端机去查 man page， 不过至少所有的分区表都能够支持哩！ ^_^

parted 可以直接在一行命令行就完成分区，是一个非常好用的指令！它常用的语法如下：

```
[root@study ~]# parted [设备] [指令 [参数]]
选项与参数：
指令功能：
          新增分区：mkpart [primary&#124;logical&#124;extended] [ext4&#124;vfat&#124;xfs] 
          显示分区：print
          删除分区：rm [partition]

范例一：以 parted 列出目前本机的分区表数据
[root@study ~]# parted /dev/vda print
Model: Virtio Block Device （virtblk）         &lt;==磁盘接口与型号
Disk /dev/vda: 42.9GB                         &lt;==磁盘文件名与容量
Sector size （logical/physical）: 512B/512B    &lt;==每个扇区的大小
Partition Table: gpt                          &lt;==是 GPT 还是 MBR 分区
Disk Flags: pmbr_boot

Number  Start   End     Size    File system     Name                  Flags
 1      1049kB  3146kB  2097kB                                        bios_grub
 2      3146kB  1077MB  1074MB  xfs
 3      1077MB  33.3GB  32.2GB                                        lvm
 4      33.3GB  34.4GB  1074MB  xfs             Linux filesystem
 5      34.4GB  35.4GB  1074MB  ext4            Microsoft basic data
 6      35.4GB  36.0GB  537MB   linux-swap（v1）  Linux swap
[  1 ]  [  2 ]  [  3  ] [  4  ] [  5  ]         [  6  ]
```

上面是最简单的 parted 指令功能简介，你可以使用“ man parted ”，或者是“ parted /dev/vda help mkpart ”去查询更详细的数据。比较有趣的地方在于分区表的输出。我们将上述的分区表示意拆成六部分来说明：

1. Number：这个就是分区的号码啦！举例来说，1号代表的是 /dev/vda1 的意思；
2. Start：分区的起始位置在这颗磁盘的多少 MB 处？有趣吧！他以容量作为单位喔！
3. End：此分区的结束位置在这颗磁盘的多少 MB 处？
4. Size：由上述两者的分析，得到这个分区有多少容量；
5. File system：分析可能的文件系统类型为何的意思！
6. Name：就如同 gdisk 的 System ID 之意。

不过 start 与 end 的单位竟然不一致！好烦～如果你想要固定单位，例如都用 MB 显示的话，可以这样做：

```
[root@study ~]# parted /dev/vda unit mb print
```

如果你想要将原本的 MBR 改成 GPT 分区表，或原本的 GPT 分区表改成 MBR 分区表，也能使用 parted ！ 但是请不要使用 vda 来测试！因为分区表格式不能转换！因此进行下面的测试后，在该磁盘的系统应该是会损毁的！ 所以鸟哥拿一颗没有使用的U盘来测试，所以文件名会变成 /dev/sda 喔！再讲一次！不要恶搞喔！

```
范例二：将 /dev/sda 这个原本的 MBR 分区表变成 GPT 分区表！（危险！危险！勿乱搞！无法复原！）
[root@study ~]# parted /dev/sda print
Model: ATA QEMU HARDDISK （scsi）
Disk /dev/sda: 2148MB
Sector size （logical/physical）: 512B/512B
Partition Table: msdos    # 确实显示的是 MBR 的 msdos 格式喔！

[root@study ~]# parted /dev/sda mklabel gpt
Warning: The existing disk label on /dev/sda will be destroyed and all data on
this disk will be lost. Do you want to continue?
Yes/No? y

[root@study ~]# parted /dev/sda print
# 你应该就会看到变成 gpt 的模样！只是...后续的分区就全部都死掉了！
```

接下来我们尝试来创建一个全新的分区吧！再次的创建一个 512MB 的分区来格式化为 vfat，且挂载于 /data/win 喔！

```
范例三：创建一个约为 512MB 容量的分区
[root@study ~]# parted /dev/vda print
.....（前面省略）.....
Number  Start   End     Size    File system     Name                  Flags
.....（中间省略）.....
 6      35.4GB  36.0GB  537MB   linux-swap（v1）  Linux swap  # 要先找出来下一个分区的起始点！

[root@study ~]# parted /dev/vda mkpart primary fat32 36.0GB 36.5GB
# 由于新的分区的起始点在前一个分区的后面，所以当然要先找出前面那个分区的 End 位置！
# 然后再请参考 mkpart 的指令功能，就能够处理好相关的动作！
[root@study ~]# parted /dev/vda print
.....（前面省略）.....
Number  Start   End     Size    File system     Name                  Flags
 7      36.0GB  36.5GB  522MB                   primary

[root@study ~]# partprobe
[root@study ~]# lsblk /dev/vda7
NAME MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda7 252:7    0  498M  0 part      # 要确定它是真的存在才行！

[root@study ~]# mkfs -t vfat /dev/vda7
[root@study ~]# blkid /dev/vda7
/dev/vda7: SEC_TYPE="msdos" UUID="6032-BF38" TYPE="vfat"

[root@study ~]# nano /etc/fstab
UUID="6032-BF38"  /data/win  vfat  defaults   0  0

[root@study ~]# mkdir /data/win
[root@study ~]# mount -a
[root@study ~]# df /data/win
Filesystem     1K-blocks  Used Available Use% Mounted on
/dev/vda7         509672     0    509672   0% /data/win
```

事实上，你应该使用 gdisk 来处理 GPT 分区就好了！不过，某些特殊时刻，例如你要自己写一只脚本，让你的分区全部一口气创建， 不需要 gdisk 一条一条指令去进行时，那么 parted 就非常有效果了！因为他可以直接进行 partition 而不需要跟用户互动！这就是它的最大好处！ 鸟哥还是建议，至少你要操作过几次 parted ，知道这家伙的用途！未来有需要再回来查！或使用 man parted 去处理喔！

7.7 重点回顾

- 一个可以被挂载的数据通常称为“文件系统, filesystem”而不是分区 （partition） 喔！
- 基本上 Linux 的传统文件系统为 Ext2 ，该文件系统内的信息主要有：
    - superblock：记录此 filesystem 的整体信息，包括inode/block的总量、使用量、剩余量， 以及文件系统的格式与相关信息等；
    - inode：记录文件的属性，一个文件占用一个inode，同时记录此文件的数据所在的 block 号码；
    - block：实际记录文件的内容，若文件太大时，会占用多个 block 。
- Ext2 文件系统的数据存取为索引式文件系统（indexed allocation）
- 需要磁盘重组的原因就是文件写入的 block 太过于离散了，此时文件读取的性能将会变的很差所致。 这个时候可以通过磁盘重组将同一个文件所属的 blocks 汇整在一起。
- Ext2文件系统主要有：boot sector, superblock, inode bitmap, block bitmap, inode table, data block 等六大部分。
- data block 是用来放置文件内容数据地方，在 Ext2 文件系统中所支持的 block 大小有 1K, 2K 及 4K 三种而已
- inode 记录文件的属性/权限等数据，其他重要项目为： 每个 inode 大小均为固定，有 128/256Bytes 两种基本容量。每个文件都仅会占用一个 inode 而已； 因此文件系统能够创建的文件数量与 inode 的数量有关；
- 文件的 block 在记录文件的实际数据，目录的 block 则在记录该目录下面文件名与其 inode 号码的对照表；
- 日志式文件系统 （journal） 会多出一块记录区，随时记载文件系统的主要活动，可加快系统复原时间；
- Linux 文件系统为增加性能，会让内存作为大量的磁盘高速缓存；
- 实体链接只是多了一个文件名对该 inode 号码的链接而已；
- 符号链接就类似Windows的捷径功能。
- 磁盘的使用必需要经过：分区、格式化与挂载，分别惯用的指令为：gdisk, mkfs, mount 三个指令
- 分区时，应使用 parted 检查分区表格式，再判断使用 fdisk/gdisk 来分区，或直接使用 parted 分区
- 为了考虑性能，XFS 文件系统格式化时，可以考虑加上 agcount/su/sw/extsize 等参数较佳
- 如果磁盘已无未分区的容量，可以考虑使用大型文件取代磁盘设备的处理方式，通过 dd 与格式化功能。
- 开机自动挂载可参考/etc/fstab之设置，设置完毕务必使用 mount -a 测试语法正确否；

7.8 本章习题 - 第一题一定要做

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ）

- 情境仿真题一：复原本章的各例题练习，本章新增非常多 partition ，请将这些 partition 删除，恢复到原本刚安装好时的状态。
  - 目标：了解到删除分区需要注意的各项信息；
  - 前提：本章的各项范例练习你都必须要做过，才会拥有 /dev/vda4 ~ /dev/vda7 出现；
  - 需求：熟悉 gdisk, parated, umount, swapoff 等指令。 由于本章处理完毕后，将会有许多新增的 partition ，所以请删除掉这两个 partition 。删除的过程需要注意的是：
  - 需先以 free / swapon -s / mount 等指令查阅，要被处理的文件系统不可以被使用！如果有被使用，则你必须要使用 umount 卸载文件系统。如果是内存交换空间，则需使用 swapon -s 找出被使用的分区， 再以 swapoff 去卸载他！

```
[root@study ~]# umount /data/ext4 /data/xfs /data/file /data/win
[root@study ~]# swapoff /dev/vda6 /tmp/swap
```

  - 观察 /etc/fstab ，该文件新增的行全部删除或注解！

```
[root@study ~]# nano /etc/fstab
.....（前面省略）.....
/dev/mapper/centos-swap swap                     swap    defaults        0 0  # 从这行
UUID="e0fa7252-b374-4a06-987a-3cb14f415488"  /data/xfs  xfs   defaults      0 0
/srv/loopdev                                 /data/file xfs   defaults,loop 0 0
UUID="6b17e4ab-9bf9-43d6-88a0-73ab47855f9d"  swap       swap  defaults      0 0
/tmp/swap                                    swap       swap  defaults      0 0
UUID="6032-BF38"                             /data/win  vfat  defaults      0 0
```

  - 使用“ gdisk /dev/vda ”删除，也可以使用“ parted /dev/vda rm 号码”删除喔！

```
[root@study ~]# parted /dev/vda rm 7
[root@study ~]# parted /dev/vda rm 6
[root@study ~]# parted /dev/vda rm 5
[root@study ~]# parted /dev/vda rm 4
[root@study ~]# partprobe
[root@study ~]# rm /tmp/swap /srv/loopdev
```

- 情境仿真题二：由于我的系统原本分区的不够好，我的用户希望能够独立一个 filesystem 附挂在 /srv/myproject 目录下。 那你该如何创建新的 filesystem ，并且让这个 filesystem 每次开机都能够自动的挂载到 /srv/myproject ， 且该目录是给 project 这个群组共享的，其他人不可具有任何权限。且该 filesystem 具有 1GB 的容量。

- 目标：理解文件系统的创建、自动挂载文件系统与专案开发必须要的权限；
- 前提：你需要进行过第六章的情境仿真才可以继续本章；
- 需求：本章的所有概念必须要清楚！ 那就让我们开始来处理这个流程吧！

- 首先，我们必须要使用 gdisk /dev/vda 来创建新的 partition。 然后按下“ n ”，按下“Enter”选择默认的分区号码，再按“Enter”选择默认的启始柱面， 按下“+1G”创建 1GB 的磁盘分区，再按下“Enter”选择默认的文件系统 ID。 可以多按一次“p ”看看是否正确，若无问题则按下“w”写入分区表；
- 避免重新开机，因此使用“ partprobe ”强制核心更新分区表；
- 创建完毕后，开始进行格式化的动作如下：“mkfs.xfs -f /dev/vda4”，这样就 OK 了！
- 开始创建挂载点，利用：“ mkdir /srv/myproject ”来创建即可；
- 编写自动挂载的配置文件：“ nano /etc/fstab ”，这个文件最下面新增一行，内容如下： /dev/vda4 /srv/myproject xfs defaults 0 0
- 测试自动挂载：“ mount -a ”，然后使用“ df /srv/myproject ”观察看看有无挂载即可！
- 设置最后的权限，使用：“ chgrp project /srv/myproject ”以及“ chmod 2770 /srv/myproject ”即可。

简答题部分：

- 我们常常说，开机的时候，“发现磁盘有问题”，请问，这个问题的产生是“filesystem 的损毁”，还是“磁盘的损毁”？ 特别需要注意的是，如果您某个 filesystem 里面，由于操作不当，可能会造成 Superblock 数据的损毁， 或者是 inode 的架构损毁，或者是 block area 的记录遗失等等，这些问题当中，其实您的“磁盘”还是好好的， 不过，在磁盘上面的“文件系统”则已经无法再利用！一般来说，我们的 Linux 很少会造成 filesystem 的损毁， 所以，发生问题时，很可能整个磁盘都损毁了。但是，如果您的主机常常不正常断电，那么， 很可能磁盘是没问题的，但是，文件系统则有损毁之虞。此时，重建文件系统（reinstall） 即可！ 不需要换掉磁盘啦！ ^_^
- 当我有两个文件，分别是 file1 与 file2 ，这两个文件互为 hard link 的文件，请问， 若我将 file1 删除，然后再以类似 vi 的方式重新创建一个名为 file1 的文件， 则 file2 的内容是否会被更动？ 这是来自网友的疑问。当我删除 file1 之后， file2 则为一个正规文件，并不会与他人共同分享同一个 inode 与 block ，因此，当我重新创建一个文件名为 file1 时，他所利用的 inode 与 block 都是由我们的 filesystem 主动去搜寻 meta data ，找到空的 inode 与 block 来创建的， 与原本的 file1 并没有任何关连性喔！所以，新建的 file1 并不会影响 file2 呢！

7.9 参考资料与延伸阅读

- [1]根据The Linux Document Project的文件所绘制的图示，详细的参考文献可以参考如下链接： Filesystem How-To: http://tldp.org/HOWTO/Filesystems-HOWTO-6.html
- [2]参考维基百科所得数据，链接网址如下： 条目 ： Ext2 介绍 http://en.wikipedia.org/wiki/Ext2
- [3]PAVE为一套秀图软件，常应用于数值模式的输出文件之再处理： PAVE 使用手册：http://www.ie.unc.edu/cempd/EDSS/pave_doc/index.shtml
- [4]详细的 inode 表格所定义的旗标可以参考如下链接： John's spec of the second extended filesystem: http://uranus.it.swin.edu.au/~jn/explore2fs/es2fs.htm
- [5]其他值得参考的 Ext2 相关文件系统文章之链接如下：
  - “Design and Implementation of the Second Extended Filesystem ”http://e2fsprogs.sourceforge.net/ext2intro.html
  - Whitepaper: Red Hat's New Journaling File System: ext3: http://www.redhat.com/support/wpapers/redhat/ext3/
  - The Second Extended File System - An introduction: http://www.freeos.com/articles/3912/
  - ext3 or ReiserFS? Hans Reiser Says Red Hat's Move Is Understandable http://www.linuxplanet.com/linuxplanet/reports/3726/1/
  - 文件系统的比较：维基百科：http://en.wikipedia.org/wiki/Comparison_of_file_systems
  - Ext2/Ext3 文件系统：http://linux.vbird.org/linux_basic/1010appendix_B.php
- [6]参考数据为：
  - man xfs 详细内容
  - xfs 官网：http://xfs.org/docs/xfsdocs-xml-dev/XFS_User_Guide/tmp/en-US/html/index.html
- [7]计算 RAID 的 sunit 与 swidth 的方式：
  - 计算 sunit 与 swidth 的方法： http://xfs.org/index.php/XFS_FAQ
  - 计算 raid 与 sunit/swidth 部落客： http://blog.tsunanet.net/2011/08/mkfsxfs-raid10-optimal-performance.html
- [8] Linux 核心所支持的硬件之设备代号（Major, Minor）查询：https://www.kernel.org/doc/Documentation/devices.txt
- [9]与 Boot sector 及 Superblock 的探讨有关的讨论文章： The Second Extended File System: http://www.nongnu.org/ext2-doc/ext2.html Rob's ext2 documentation: http://www.landley.net/code/toybox/ext2.html

2002/07/15：第一次完成 2003/02/07：重新编排与加入 FAQ 2004/03/15：修改 inode 的说明，并且将链接文件的说明移动至这个章节当中！ 2005/07/20：将旧的文章移动到 这里 。2005/07/22：将原本的附录一与附录二移动成为附录 B 啦！ 2005/07/26：做了一个比较完整

的修订，加入较完整的 ext3 的说明～ 2005/09/08：看到了一篇讨论，说明 FC4 在默认的环境中，使用 mkswap 会有问题。 2005/10/11：新增加了一个目录的 link 数量说明！ 2005/11/11：增加了一个 fsck 的 -f 参数在里头！ 2006/03/02：参考：这里的说明，将 ext2/ext3 最大文件系统由 16TB 改为 32TB。 2006/03/31：增加了虚拟内存的相关说明. 2006/05/01：将磁盘扇区的图做个修正，感谢网友 LiaoLiang 兄提供的信息！并加入参考文献！ 2006/06/09：增加 hard link 不能链接到目录的原因，详情参考：http://phorum.study-area.org/viewtopic.php?t=12235 2006/06/28：增加关于 loop device 的相关说明呐！ 2006/09/08：加入 mknod 内的设备代号说明 ，以及列出 Linux 核心网站的设备代号查询。 2008/09/29：原本的FC4系列文章移动到此处 2008/10/24：由于软盘的使用已经越来越少了，所以将 fdformat 及 mkbootdisk 拿掉了！ 2008/10/31：这个月事情好多～花了一个月才将数据整理完毕！修改幅度非常的大喔！ 2008/11/01：最后一节的利用 GNU 的 parted 进行分区行为误植为 GUN ！感谢网友阿贤的来信告知！ 2008/12/05：感谢网友 ian_chen 的告知，之前将 flash 当成 flush 了！真抱歉！已更新！ 2009/04/01：感谢讨论区网友提供的说明， 鸟哥之前 superblock 这里写得不够好，有订正说明，请帮忙看看。 2009/08/19：加入两题情境仿真，重新修订一题简答题。 2009/08/30：加入 du 的 -S 说明中。 2015/06/17：将旧的基于 CentOS5 的版本移动到这里。 2015/10/26：加上 noexec 的额外说明！感谢网友在讨论区的建议！

第八章、文件与文件系统的压缩,打包与备份

最近更新日期：20//

在 Linux 下面有相当多的压缩指令可以运行喔！这些压缩指令可以让我们更方便从网络上面下载容量较大的文件呢！ 此外，我们知道在 Linux 下面的扩展名是没有什么很特殊的意义的，不过，针对这些压缩指令所做出来的压缩文件， 为了方便记忆，还是会有一些特殊的命名方式啦！就让我们来看看吧！

8.1 压缩文件的用途与技术

你是否有过文件文件太大，导致无法以正常的 email 方式发送出去 （很多 email 都有容量大约 25MB 每封信的限制啊！）？ 又或者学校、厂商要求使用 CD 或 DVD 来传递归档用的数据，但是你的单一文件却都比这些传统的一次性储存媒体还要大！那怎么分成多片来烧录呢？ 还有，你是否有过要备份某些重要数据，偏偏这些数据量太大了，耗掉了你很多的磁盘空间呢？ 这个时候，那个好用的“文件压缩”技术可就派的上用场了！

因为这些比较大型的文件通过所谓的文件压缩技术之后，可以将他的磁盘使用量降低，可以达到减低文件大小的效果。此外，有的压缩程序还可以进行容量限制， 使一个大型文件可以分区成为数个小型文件，以方便软盘片携带呢！

那么什么是“文件压缩”呢？我们来稍微谈一谈他的原理好了。目前我们使用的计算机系统中都是使用所谓的 Bytes 单位来计量的！不过，事实上，计算机最小的计量单位应该是 bits 才对啊。此外，我们也知道 1 Byte = 8 bits 。但是如果今天我们只是记忆一个数字，亦即是 1 这个数字呢？他会如何记录？假设一个 Byte 可以看成下面的模样：

> □□□□□□□□□



Tips 由于 1 Byte = 8 bits ，所以每个 Byte 当中会有 8 个空格，而每个空格可以是 0, 1 ，这里仅是做为一个约略的介绍， 更多的详细数据请参考第零章的计算机概论吧！

由于我们记录数字是 1 ，考虑计算机所谓的二进制喔，如此一来， 1 会在最右边占据 1 个 bit ，而其他的 7 个 bits 将会自动的被填上 0 啰！你看看，其实在这样的例子中，那 7 个 bits 应该是“空的”才对！不过，为了要满足目前我们的操作系统数据的存取，所以就会将该数据转为 Byte 的型态来记录了！而一些聪明的计算机工程师就利用一些复杂的计算方式， 将这些没有使用到的空间“丢”出来，以让文件占用的空间变小！这就是压缩的技术啦！

另外一种压缩技术也很有趣，他是将重复的数据进行统计记录的。举例来说，如果你的数据为“111....”共有100个1时， 那么压缩技术会记录为“100个1”而不是真的有100个1的位存在！这样也能够精简文件记录的容量呢！ 非常有趣吧！

简单的说，你可以将他想成，其实文件里面有相当多的“空间”存在，并不是完全填满的，而“压缩”的技术就是将这些“空间”填满，以让整个文件占用的容量下降！ 不过，这些“压缩过的文件”并无法直接被我们的操作系统所使用的，因此， 若要使用这些被压缩过的文件数据，

则必须将他“还原”回来未压缩前的模样， 那就是所谓的“解压缩”啰！而至于压缩后与压缩的文件所占用的磁盘空间大小， 就可以被称为是“压缩比”啰！更多的技术文件或许你可以参考一下：

- RFC 1952 文件：http://www.ietf.org/rfc/rfc1952.txt
- 鸟哥站上的备份：http://linux.vbird.org/linux_basic/0240tarcompress/0240tarcompress_gzip.php

这个“压缩”与“解压缩”的动作有什么好处呢？最大的好处就是压缩过的文件大小变小了， 所以你的硬盘容量无形之中就可以容纳更多的数据。此外，在一些网络数据的传输中，也会由于数据量的降低， 好让网络带宽可以用来作更多的工作！而不是老是卡在一些大型的文件传输上面呢！目前很多的 WWW 网站也是利用文件压缩的技术来进行数据的传送，好让网站带宽的可利用率上升喔！



Tips 上述的WWW网站压缩技术蛮有趣的！他让你网站上面“看的到的数据”在经过网络传输时，使用的是“压缩过的数据”， 等到这些压缩过的数据到达你的计算机主机时，再进行解压缩，由于目前的计算机运算速度相当的快速， 因此其实在网页浏览的时候，时间都是花在“数据的传输”上面，而不是 CPU 的运算啦！如此一来，由于压缩过的数据量降低了，自然传送的速度就会增快不少！

若你是一位软件工程师，那么相信你也会喜欢将你自己的软件压缩之后提供大家下载来使用， 毕竟没有人喜欢自己的网站天天都是带宽满载的吧？举个例子来说， Linux 3.10.81（CentOS 7 用的延伸版本） 完整的核心大小约有 570 MB 左右，而由于核心主要多是 ASCII code 的纯文本体态文件，这种文件的“多余空间”最多了。而一个提供下载的压缩过的 3.10.81 核心大约仅有 76MB 左右，差了几倍呢？你可以自己算一算喔！

8.2 Linux 系统常见的压缩指令

在Linux的环境中，压缩文件的扩展名大多是："*.tar, *.tar.gz, *.tgz, *.gz, *.Z, *.bz2, *.xz"，为什么会有这样的扩展名呢？不是说 Linux 的扩展名没有什么作用吗？

这是因为 Linux 支持的压缩指令非常多，且不同的指令所用的压缩技术并不相同，当然彼此之间可能就无法互通压缩/解压缩文件啰。 所以，当你下载到某个压缩文件时，自然就需要知道该文件是由哪种压缩指令所制作出来的，好用来对照着解压缩啊！ 也就是说，虽然 Linux 文件的属性基本上是与文件名没有绝对关系的， 但是为了帮助我们人类小小的脑袋瓜子，所以适当的扩展名还是必要的！ 下面我们就列出几个常见的压缩文件扩展名吧：

```
*.Z         compress 程序压缩的文件；
*.zip       zip 程序压缩的文件；
*.gz        gzip 程序压缩的文件；
*.bz2       bzip2 程序压缩的文件；
*.xz        xz 程序压缩的文件；
*.tar       tar 程序打包的数据，并没有压缩过；
*.tar.gz    tar 程序打包的文件，其中并且经过 gzip 的压缩
*.tar.bz2   tar 程序打包的文件，其中并且经过 bzip2 的压缩
*.tar.xz    tar 程序打包的文件，其中并且经过 xz 的压缩
```

Linux上常见的压缩指令就是 gzip, bzip2 以及最新的 xz ，至于 compress 已经退流行了。为了支持 windows 常见的 zip，其实 Linux 也早就有 zip 指令了！ gzip 是由 GNU 计划所开发出来的压缩指令，该指令已经取代了 compress 。 后来 GNU 又开发出 bzip2 及 xz 这几个压缩比更好的压缩指令！不过，这些指令通常仅能针对一个文件来压缩与解压缩，如此一来， 每次压缩与解压缩都要一大堆文件，岂不烦人？此时，那个所谓的"打包软件, tar"就显的很重要啦！

这个 tar 可以将很多文件"打包"成为一个文件！甚至是目录也可以这么玩。不过，单纯的 tar 功能仅是"打包"而已，亦即是将很多文件集结成为一个文件， 事实上，他并没有提供压缩的功能，后来，GNU 计划中，将整个 tar 与压缩的功能结合在一起，如此一来提供使用者更方便并且更强大的压缩与打包功能！ 下面我们就来谈一谈这些在 Linux 下面基本的压缩指令吧！

8.2.1 gzip, zcat/zmore/zless/zgrep

gzip 可以说是应用度最广的压缩指令了！目前 gzip 可以解开 compress, zip 与 gzip 等软件所压缩的文件。 至于 gzip 所创建的压缩文件为 *.gz 的文件名喔！让我们来看看这个指令的语法吧：

```
[dmtsai@study ~]$ gzip [-cdtv#] 文件名
[dmtsai@study ~]$ zcat 文件名.gz
选项与参数：
-c  ：将压缩的数据输出到屏幕上，可通过数据流重导向来处理；
-d  ：解压缩的参数；
-t  ：可以用来检验一个压缩文件的一致性～看看文件有无错误；
-v  ：可以显示出原文件/压缩文件的压缩比等信息；
-#  ：# 为数字的意思，代表压缩等级，-1 最快，但是压缩比最差、-9 最慢，但是压缩比最好！默认是 -6

范例一：找出 /etc 下面 （不含子目录） 容量最大的文件，并将它复制到 /tmp ，然后以 gzip 压缩
[dmtsai@study ~]$ ls -ldSr /etc/*   # 忘记选项意义？请自行 man 啰！
.....（前面省略）.....
-rw-r--r--.  1 root root    25213 Jun 10  2014 /etc/dnsmasq.conf
-rw-r--r--.  1 root root    69768 May  4 17:55 /etc/ld.so.cache
-rw-r--r--.  1 root root   670293 Jun  7  2013 /etc/services

[dmtsai@study ~]$ cd /tmp
[dmtsai@study tmp]$ cp /etc/services .
[dmtsai@study tmp]$ gzip -v services
services:        79.7% -- replaced with services.gz
[dmtsai@study tmp]$ ll /etc/services /tmp/services*
-rw-r--r--. 1 root   root   670293 Jun  7  2013 /etc/services
-rw-r--r--. 1 dmtsai dmtsai 136088 Jun 30 18:40 /tmp/services.gz
```

当你使用 gzip 进行压缩时，在默认的状态下原本的文件会被压缩成为 .gz 的文件名，原始文件就不再存在了。 这点与一般习惯使用 windows 做压缩的朋友所熟悉的情况不同喔！要注意！要注意！ 此外，使用 gzip 压缩的文件在 Windows 系统中，竟然可以被 WinRAR/7zip 这个软件解压缩呢！很好用吧！至于其他的用法如下：

```
范例二：由于 services 是文本文件，请将范例一的压缩文件的内容读出来！
[dmtsai@study tmp]$ zcat services.gz
# 由于 services 这个原本的文件是是文本文件，因此我们可以尝试使用 zcat/zmore/zless 去读取！
# 此时屏幕上会显示 servcies.gz 解压缩之后的原始文件内容！

范例三：将范例一的文件解压缩
[dmtsai@study tmp]$ gzip -d services.gz
# 鸟哥不要使用 gunzip 这个指令，不好背！使用 gzip -d 来进行解压缩！
# 与 gzip 相反， gzip -d 会将原本的 .gz 删除，回复到原本的 services 文件。

范例四：将范例三解开的 services 用最佳的压缩比压缩，并保留原本的文件
[dmtsai@study tmp]$ gzip -9 -c services &gt; services.gz

范例五：由范例四再次创建的 services.gz 中，找出 http 这个关键字在哪几行？
[dmtsai@study tmp]$ zgrep -n 'http' services.gz
14:#       http://www.iana.org/assignments/port-numbers
89:http            80/tcp          www www-http    # WorldWideWeb HTTP
90:http            80/udp          www www-http    # HyperText Transfer Protocol
.....（下面省略）.....
```

其实 gzip 的压缩已经最优化过了，所以虽然 gzip 提供 1~9 的压缩等级，不过使用默认的 6 就非常好用了！ 因此上述的范例四可以不要加入那个 -9 的选项。范例四的重点在那个 -c 与 > 的使用啰！-c 可以将原本要转成压缩文件的数据内容，将它变成文字类型从屏幕输出， 然后我们可以通过大于 （>） 这个符号，将原本应该由屏幕输出的数据，转成输出到文件而不是屏幕，所以就能够创建出压缩挡了。只是文件名也要自己写， 当然最好还是遵循 gzip 的压缩文件名要求较佳喔！！更多的 > 这个符号的应用，我们会在 bash 章节再次提及！

cat/more/less 可以使用不同的方式来读取纯文本文件，那个 zcat/zmore/zless 则可以对应于 cat/more/less 的方式来读取纯文本文件被压缩后的压缩文件！ 由于 gzip 这个压缩指令主要想要用来取代 compress 的，所以不但 compress 的压缩文件可以使用 gzip 来解开，同时 zcat 这个指令可以同时读取 compress 与 gzip 的压缩文件呦！

另外，如果你还想要从文字压缩文件当中找数据的话，可以通过 egrep 来搜寻关键字喔！而不需要将压缩文件解开才以 grep 进行！ 这对查询备份中的文本文件数据相当有用！



Tips 时至今日，应该也没有人爱用 compress 这个老老的指令了！因此，这一章已经拿掉了 compress 的介绍～而如果你还有备份数据使用的是 compress 创建出来的 .Z 文件，那也无须担心，使用 znew 可以将该文件转成 gzip 的格示喔！

8.2.2 bzip2, bzcat/bzmore/bzless/bzgrep

若说 gzip 是为了取代 compress 并提供更好的压缩比而成立的，那么 bzip2 则是为了取代 gzip 并提供更佳的压缩比而来的。 bzip2 真是很不错用的东西～这玩意的压缩比竟然比 gzip 还要好～至于 bzip2 的用法几乎与 gzip 相同！看看下面的用法吧！

```
[dmtsai@study ~]$ bzip2 [-cdkzv#] 文件名
[dmtsai@study ~]$ bzcat 文件名.bz2
选项与参数：
-c  ：将压缩的过程产生的数据输出到屏幕上！
-d  ：解压缩的参数
-k  ：保留原始文件，而不会删除原始的文件喔！
-z  ：压缩的参数 （默认值，可以不加）
-v  ：可以显示出原文件/压缩文件的压缩比等信息；
-#  ：与 gzip 同样的，都是在计算压缩比的参数， -9 最佳， -1 最快！

范例一：将刚刚 gzip 范例留下来的 /tmp/services 以 bzip2 压缩
[dmtsai@study tmp]$ bzip2 -v services
  services:  5.409:1,  1.479 bits/Byte, 81.51% saved, 670293 in, 123932 out.
[dmtsai@study tmp]$ ls -l services*
-rw-r--r--. 1 dmtsai dmtsai 123932 Jun 30 18:40 services.bz2
-rw-rw-r--. 1 dmtsai dmtsai 135489 Jun 30 18:46 services.gz
# 此时 services 会变成 services.bz2 之外，你也可以发现 bzip2 的压缩比要较 gzip 好喔！！
# 压缩率由 gzip 的 79% 提升到 bzip2 的 81% 哩！

范例二：将范例一的文件内容读出来！
[dmtsai@study tmp]$ bzcat services.bz2

范例三：将范例一的文件解压缩
[dmtsai@study tmp]$ bzip2 -d services.bz2

范例四：将范例三解开的 services 用最佳的压缩比压缩，并保留原本的文件
[dmtsai@study tmp]$ bzip2 -9 -c services &gt; services.bz2
```

看上面的范例，你会发现到 bzip2 连选项与参数都跟 gzip 一模一样！只是扩展名由 .gz 变成 .bz2 而已！其他的用法都大同小异，所以鸟哥就不一一介绍了！ 你也可以发现到 bzip2 的压缩率确实比 gzip 要好些！不过，对于大容量文件来说，bzip2 压缩时间会花比较久喔！至少比 gzip 要久的多！ 这没办法～要有更多可用容量，就得要花费相对应的时间！还 OK 啊！

8.2.3 xz, xzcat/xzmore/xzless/xzgrep

虽然 bzip2 已经具有很棒的压缩比，不过显然某些自由软件开发者还不满足，因此后来还推出了 xz 这个压缩比更高的软件！这个软件的用法也跟 gzip/bzip2 几乎一模一样！ 那我们就来瞧一瞧！

```
[dmtsai@study ~]$ xz [-dtlkc#] 文件名
[dmtsai@study ~]$ xcat 文件名.xz
选项与参数：
-d  ：就是解压缩啊！
-t  ：测试压缩文件的完整性，看有没有错误
-l  ：列出压缩文件的相关信息
-k  ：保留原本的文件不删除～
-c  ：同样的，就是将数据由屏幕上输出的意思！
-#  ：同样的，也有较佳的压缩比的意思！

范例一：将刚刚由 bzip2 所遗留下来的 /tmp/services 通过 xz 来压缩！
[dmtsai@study tmp]$ xz -v services
services (1/1)
  100 %        97.3 KiB / 654.6 KiB = 0.149

[dmtsai@study tmp]$ ls -l services*
-rw-rw-r--. 1 dmtsai dmtsai 123932 Jun 30 19:09 services.bz2
-rw-rw-r--. 1 dmtsai dmtsai 135489 Jun 30 18:46 services.gz
-rw-r--r--. 1 dmtsai dmtsai  99608 Jun 30 18:40 services.xz
# 各位观众！看到没有啊！！容量又进一步下降的更多耶！好棒的压缩比！

范例二：列出这个压缩文件的信息，然后读出这个压缩文件的内容
[dmtsai@study tmp]$ xz -l services.xz
Strms  Blocks   Compressed Uncompressed  Ratio  Check   Filename
    1       1      97.3 KiB    654.6 KiB  0.149  CRC64   services.xz
# 竟然可以列出这个文件的压缩前后的容量，真是太人性化了！这样观察就方便多了！

[dmtsai@study tmp]$ xzcat services.xz

范例三：将他解压缩吧！
[dmtsai@study tmp]$ xz -d services.xz

范例四：保留原文件的文件名，并且创建压缩文件！
[dmtsai@study tmp]$ xz -k services
```

虽然 xz 这个压缩比真的好太多太多了！以鸟哥选择的这个 services 文件为范例，他可以将 gzip 压缩比 （压缩后/压缩前） 的 21% 更进一步优化到 15% 耶！ 差非常非常多！不过， xz 最大的问题是...时间花太久了！如果你曾经使用过 xz 的话，应该会有发现，他的运算时间真的比 gzip 久很多喔！

鸟哥以自己的系统，通过“ time [gzip|bzip2|xz] -c services > services.[gz|bz2|xz] ”去执行运算结果，结果发现这三个指令的执行时间依序是： 0.019s, 0.042s, 0.261s， 看最后一个数字！差了 10 倍的时间耶！所以，如果你并不觉得时间是你的成本考虑，那么使用 xz 会比较好！如果时间是你的重要成本，那么 gzip 恐怕是比较适合的压缩软件喔！

8.3 打包指令： tar

前一小节谈到的指令大多仅能针对单一文件来进行压缩，虽然 gzip, bzip2, xz 也能够针对目录来进行压缩，不过， 这两个指令对目录的压缩指的是“将目录内的所有文件 "分别" 进行压缩”的动作！而不像在 Windows 的系统，可以使用类似 WinRAR 这一类的压缩软件来将好多数据“包成一个文件”的样式。

这种将多个文件或目录包成一个大文件的指令功能，我们可以称呼他是一种“打包指令”啦！ 那 Linux 有没有这种打包指令呢？是有的！那就是鼎鼎大名的 tar 这个玩意儿了！ tar 可以将多个目录或文件打包成一个大文件，同时还可以通过 gzip/bzip2/xz 的支持，将该文件同时进行压缩！ 更有趣的是，由于 tar 的使用太广泛了，目前 Windows 的 WinRAR 也支持 .tar.gz 文件名的解压缩呢！ 很不错吧！所以下面我们就来玩一玩这个咚咚！

8.3.1 tar

tar 的选项与参数非常的多！我们只讲几个常用的选项，更多选项您可以自行 man tar 查询啰！

```
[dmtsai@study ~]$ tar [-z&#124;-j&#124;-J] [cv] [-f 待创建的新文件名] filename... &lt;==打包
[dmtsai@study ~]$ tar [-z&#124;-j&#124;-J] [tv] [-f 既有的 tar文件名]             &lt;==察看
[dmtsai@study ~]$ tar [-z&#124;-j&#124;-J] [xv] [-f 既有的 tar文件名] [-C 目录]   &lt;==解压
选项与参数：
-c  ：创建打包文件，可搭配 -v 来察看过程中被打包的文件名（filename）
-t  ：察看打包文件的内容含有哪些文件名，重点在察看“文件名”就是了；
-x  ：解打包或解压缩的功能，可以搭配 -C （大写） 在特定目录解开
      特别留意的是， -c, -t, -x 不可同时出现在一串命令行中。
-z  ：通过 gzip  的支持进行压缩/解压缩：此时文件名最好为 *.tar.gz
-j  ：通过 bzip2 的支持进行压缩/解压缩：此时文件名最好为 *.tar.bz2
-J  ：通过 xz    的支持进行压缩/解压缩：此时文件名最好为 *.tar.xz
      特别留意， -z, -j, -J 不可以同时出现在一串命令行中
-v  ：在压缩/解压缩的过程中，将正在处理的文件名显示出来！
-f filename：-f 后面要立刻接要被处理的文件名！建议 -f 单独写一个选项啰！（比较不会忘记）
-C 目录    ：这个选项用在解压缩，若要在特定目录解压缩，可以使用这个选项。

其他后续练习会使用到的选项介绍：
-p（小写） ：保留备份数据的原本权限与属性，常用于备份（-c）重要的配置文件
-P（大写） ：保留绝对路径，亦即允许备份数据中含有根目录存在之意；
--exclude=FILE：在压缩的过程中，不要将 FILE 打包！
```

其实最简单的使用 tar 就只要记忆下面的方式即可：

- 压　缩：tar -j<u>c</u>v -f filename.tar.bz2 要被压缩的文件或目录名称
- 查　询：tar -j<u>t</u>v -f filename.tar.bz2
- 解压缩：tar -j<u>x</u>v -f filename.tar.bz2 -C 欲解压缩的目录

那个 filename.tar.bz2 是我们自己取的文件名，tar 并不会主动的产生创建的文件名喔！我们要自订啦！ 所以扩展名就显的很重要了！如果不加 [-z|-j|-J] 的话，文件名最好取为 *.tar* 即可。如果是 *-j* 选项，代表有 *bzip2* 的支持，因此文件名最好就取为 .tar.bz2 ，因为 bzip2 会产

生 .bz2 的扩展名之故！ 至于如果是加上了 -z 的 gzip 的支持，那文件名最好取为 *.tar.gz 喔！了解乎？

另外，由于“ -f filename ”是紧接在一起的，过去很多文章常会写成“-jcvf filename”，这样是对的， 但由于选项的顺序理论上是可以变换的，所以很多读者会误认为“-jvfc filename”也可以～事实上这样会导致产生的文件名变成 c ！ 因为 -fc 嘛！所以啰，建议您在学习 tar 时，将“ -f filename ”与其他选项独立出来，会比较不容易发生问题。

闲话少说，让我们来测试几个常用的 tar 方法吧！

- 使用 tar 加入 -z, -j 或 -J 的参数备份 /etc/ 目录

有事没事备份一下 /etc 这个目录是件好事！备份 /etc 最简单的方法就是使用 tar 啰！让我们来玩玩先：

```
[dmtsai@study ~]$ su -  # 因为备份 /etc 需要 root 的权限，否则会出现一堆错误
[root@study ~]# time tar -zpcv -f /root/etc.tar.gz /etc
tar: Removing leading `/' from member names  &lt;==注意这个警告讯息
/etc/
....（中间省略）....
/etc/hostname
/etc/aliases.db

real    0m0.799s   # 多了 time 会显示程序运行的时间！看 real 就好了！花去了 0.799s
user    0m0.767s
sys     0m0.046s
# 由于加上 -v 这个选项，因此正在作用中的文件名就会显示在屏幕上。
# 如果你可以翻到第一页，会发现出现上面的错误讯息！下面会讲解。
# 至于 -p 的选项，重点在于“保留原本文件的权限与属性”之意。

[root@study ~]# time tar -jpcv -f /root/etc.tar.bz2 /etc
....（前面省略）....
real    0m1.913s
user    0m1.881s
sys     0m0.038s
[root@study ~]# time tar -Jpcv -f /root/etc.tar.xz  /etc
....（前面省略）....
real    0m9.023s
user    0m8.984s
sys     0m0.086s
# 显示的讯息会跟上面一模一样啰！不过时间会花比较多！使用了 -J 时，会花更多时间

[root@study ~]# ll /root/etc*
-rw-r--r--. 1 root root 6721809 Jul  1 00:16 /root/etc.tar.bz2
-rw-r--r--. 1 root root 7758826 Jul  1 00:14 /root/etc.tar.gz
-rw-r--r--. 1 root root 5511500 Jul  1 00:16 /root/etc.tar.xz
[root@study ~]# du -sm /etc
28      /etc  # 实际目录约占有 28MB 的意思！
```

压缩比越好当然要花费的运算时间越多！我们从上面可以看到，虽然使用 gzip 的速度相当快，总时间花费不到 1 秒钟，但是压缩率最糟糕！ 如果使用 xz 的话，虽然压缩比最佳！不过竟然花了 9 秒钟的时间耶！这还仅是备份 28MBytes 的 /etc 而已，如果备份的数据是很大容量的， 那你真的要考虑时间成本才行！

至于加上“ -p ”这个选项的原因是为了保存原本文件的权限与属性！我们曾在第六章的 cp 指令介绍时谈到权限与文件类型（例如链接文件）对复制的不同影响。 同样的，在备份重要的系统数据时，这些原本文件的权限需要做完整的备份比较好。此时 -p 这个选项就派的上用场

了。 接下来让我们看看打包文件内有什么数据存在？

- 查阅 tar 文件的数据内容 （可察看文件名），与备份文件名有否根目录的意义

要察看由 tar 所创建的打包文件内部的文件名非常的简单！可以这样做：

```
[root@study ~]# tar -jtv -f /root/etc.tar.bz2
....（前面省略）....
-rw-r--r-- root/root       131 2015-05-25 17:48 etc/locale.conf
-rw-r--r-- root/root        19 2015-05-04 17:56 etc/hostname
-rw-r--r-- root/root     12288 2015-05-04 17:59 etc/aliases.db
```

如果加上 -v 这个选项时，详细的文件权限/属性都会被列出来！如果只是想要知道文件名而已， 那么就将 -v 拿掉即可。从上面的数据我们可以发现一件很有趣的事情，那就是每个文件名都没了根目录了！这也是上一个练习中出现的那个警告讯息“tar: Removing leading `/' from member names（移除了文件名开头的` /'）”所告知的情况！

那为什么要拿掉根目录呢？主要是为了安全！我们使用 tar 备份的数据可能会需要解压缩回来使用， 在 tar 所记录的文件名 （就是我们刚刚使用 tar -jtvf 所察看到的文件名） 那就是解压缩后的实际文件名。 如果拿掉了根目录，假设你将备份数据在 /tmp 解开，那么解压缩的文件名就会变成“/tmp/<u>etc/xxx</u>”。 但“如果没有拿掉根目录，解压缩后的文件名就会是绝对路径， 亦即解压缩后的数据一定会被放置到 <u>/etc/xxx</u> 去！”如此一来，你的原本的 /etc/ 下面的数据， 就会被备份数据所覆盖过去了！



Tips 你会说：“既然是备份数据，那么还原回来也没有什么问题吧？”想像一个状况，你备份的数据是两年前的旧版 CentOS 6.x， 你只是想要了解一下过去的备份内容究竟有哪些数据而已，结果一解开该文件，却发现你目前新版的 CentOS 7.x 下面的 /etc 被旧版的备份数据覆盖了！此时你该如何是好？大概除了哭哭你也不能做啥事吧？所以啰，当然是拿掉根目录比较安全一些的。

如果你确定你就是需要备份根目录到 tar 的文件中，那可以使用 -P （大写） 这个选项，请看下面的例子分析：

```
范例：将文件名中的（根）目录也备份下来，并察看一下备份文件的内容文件名
[root@study ~]# tar -jp&lt;u&gt;P&lt;/u&gt;cv -f /root/etc.and.root.tar.bz2 /etc

[root@study ~]# tar -jtf /root/etc.and.root.tar.bz2
/etc/locale.conf
/etc/hostname
/etc/aliases.db
# 这次查阅文件名不含 -v 选项，所以仅有文件名而已！没有详细属性/权限等参数。
```

有发现不同点了吧？如果加上 -P 选项，那么文件名内的根目录就会存在喔！不过，鸟哥个人建议，还是不要加上 -P 这个选项来备份！ 毕竟很多时候，我们备份是为了要未来追踪问题用的，倒不一定需要还原回原本的系统中！ 所以拿掉根目录后，备份数据的应用会比较有弹性！也比较安全呢！

- 将备份的数据解压缩，并考虑特定目录的解压缩动作 （-C 选项的应用）

那如果想要解打包呢？很简单的动作就是直接进行解打包嘛！

```
[root@study ~]# tar -jxv -f /root/etc.tar.bz2
[root@study ~]# ll
....（前面省略）....
drwxr-xr-x. 131 root root    8192 Jun 26 22:14 etc
....（后面省略）....
```

此时该打包文件会在“本目录下进行解压缩”的动作！ 所以，你等一下就会在主文件夹下面发现一个名为 etc 的目录啰！所以啰，如果你想要将该文件在 /tmp 下面解开， 可以 cd /tmp 后，再下达上述的指令即可。不过，这样好像很麻烦呢～有没有更简单的方法可以“指定欲解开的目录”呢？ 有的，可以使用 -C 这个选项喔！举例来说：

```
[root@study ~]# tar -jxv -f /root/etc.tar.bz2 -C /tmp
[root@study ~]# ll /tmp
....（前面省略）....
drwxr-xr-x. 131 root   root     8192 Jun 26 22:14 etc
....（后面省略）....
```

这样一来，你就能够将该文件在不同的目录解开啰！鸟哥个人是认为，这个 -C 的选项务必要记忆一下的！ 好了，处理完毕后，请记得将这两个目录删除一下呢！

```
[root@study ~]# rm -rf /root/etc /tmp/etc
```

再次强调，这个“ rm -rf ”是很危险的指令！下达时请务必要确认一下后面接的文件名。我们要删除的是 /root/etc 与 /tmp/etc， 您可不要将 /etc/ 删除掉了！系统会死掉的～ ^_^

- 仅解开单一文件的方法

刚刚上头我们解压缩都是将整个打包文件的内容全部解开！想像一个情况，如果我只想要解开打包文件内的其中一个文件而已， 那该如何做呢？很简单的，你只要使用 -jtv 找到你要的文件名，然后将该文件名解开即可。 我们用下面的例子来说明一下：

```
# 1\. 先找到我们要的文件名，假设解开 shadow 文件好了：
[root@study ~]# tar -jtv -f /root/etc.tar.bz2 &#124; grep 'shadow'
---------- root/root       721 2015-06-17 00:20 etc/gshadow
---------- root/root      1183 2015-06-17 00:20 etc/shadow-
---------- root/root      1210 2015-06-17 00:20 etc/shadow  &lt;==这是我们要的！
---------- root/root       707 2015-06-17 00:20 etc/gshadow-
# 先搜寻重要的文件名！其中那个 grep 是“撷取”关键字的功能！我们会在第三篇说明！
# 这里您先有个概念即可！那个管线 &#124; 配合 grep 可以撷取关键字的意思！

# 2\. 将该文件解开！语法与实际作法如下：
[root@study ~]# tar -jxv -f 打包档.tar.bz2 待解开文件名
[root@study ~]# tar -jxv -f /root/etc.tar.bz2 etc/shadow
etc/shadow
[root@study ~]# ll etc
total 4
----------. 1 root root 1210 Jun 17 00:20 shadow
# 很有趣！此时只会解开一个文件而已！不过，重点是那个文件名！你要找到正确的文件名。
# 在本例中，你不能写成 /etc/shadow ！因为记录在 etc.tar.bz2 内的并没有 / 之故！
```



Tips 在这个练习之前，你可能要先将前面练习所产生的 /root/etc 删除才行！不然 /root/etc/shadow 会重复存在，而其他的前面实验的文件也会存在，那就看不出什么鬼～

- 打包某目录，但不含该目录下的某些文件之作法

假设我们想要打包 /etc/ /root 这几个重要的目录，但却不想要打包 /root/etc* 开头的文件，因为该文件都是刚刚我们才创建的备份文件嘛！而且假设这个新的打包文件要放置成为 /root/system.tar.bz2 ，当然这个文件自己不要打包自己 （因为这个文件放置在 /root 下面啊！），此时我们可以通过 --exclude 的帮忙！那个 exclude 就是不包含的意思！所以你可以这样做：

```
[root@study ~]# tar -jcv  -f /root/system.tar.bz2 --exclude=/root/etc* \
&gt; --exclude=/root/system.tar.bz2  /etc /root
```

上面的指令是一整列的～其实你可以打成：“tar -jcv -f /root/system.tar.bz2 --exclude=/root/etc* --exclude=/root/system.tar.bz2 /etc /root”，如果想要两行输入时，最后面加上反斜线 （\） 并立刻按下 [enter] ， 就能够到第二行继续输入了。这个指令下达的方式我们会在第三章再仔细说明。 通过这个 --exclude="file" 的动作，我们可以将几个特殊的文件或目录移除在打包之列，让打包的动作变的更简便喔！^_^

- 仅备份比某个时刻还要新的文件

某些情况下你会想要备份新的文件而已，并不想要备份旧文件！此时 --newer-mtime 这个选项就粉重要啦！ 其实有两个选项啦，一个是“ --newer ”另一个就是“ --newer-mtime ”，这两个选项有何不同呢？ 我们在 第六章的 touch 介绍中谈到过三种不同的时间参数， 当使用 --

newer 时，表示后续的日期包含“ mtime 与 ctime ”，而 --newer-mtime 则仅是 mtime 而已！这样知道了吧！ ^_^ 。那就让我们来尝试处理一下啰！

```
# 1\. 先由 [find](../Text/index.html#find) 找出比 /etc/passwd 还要新的文件
[root@study ~]# find /etc -newer /etc/passwd
....（过程省略）....
# 此时会显示出比 /etc/passwd 这个文件的 mtime 还要新的文件名，
# 这个结果在每部主机都不相同！您先自行查阅自己的主机即可，不会跟鸟哥一样！

[root@study ~]# ll /etc/passwd
-rw-r--r--. 1 root root 2092  Jun 17 00:20 /etc/passwd

# 2\. 好了，那么使用 tar 来进行打包吧！日期为上面看到的 2015/06/17
[root@study ~]# tar -jcv -f /root/etc.newer.then.passwd.tar.bz2 \
&gt; --newer-mtime="2015/06/17" /etc/*
tar: Option --newer-mtime: Treating date `2015/06/17' as 2015-06-17 00:00:00
tar: Removing leading `/' from member names
/etc/abrt/
....（中间省略）....
/etc/alsa/
/etc/yum.repos.d/
....（中间省略）....
tar: /etc/yum.repos.d/CentOS-fasttrack.repo: file is unchanged; not dumped
# 最后行显示的是“没有被备份的”，亦即 not dumped 的意思！

# 3\. 显示出文件即可
[root@study ~]# tar -jtv -f /root/etc.newer.then.passwd.tar.bz2 &#124; grep -v '/$'
# 通过这个指令可以调用出 tar.bz2 内的结尾非 / 的文件名！就是我们要的啦！
```

现在你知道这个指令的好用了吧！甚至可以进行差异文件的记录与备份呢～ 这样子的备份就会显的更容易啰！你可以这样想像，如果我在一个月前才进行过一次完整的数据备份， 那么这个月想要备份时，当然可以仅备份上个月进行备份的那个时间点之后的更新的文件即可！为什么呢？因为原本的文件已经有备份了嘛！干嘛还要进行一次？只要备份新数据即可。这样可以降低备份的容量啊！

- 基本名称： tarfile, tarball ？

另外值得一提的是，tar 打包出来的文件有没有进行压缩所得到文件称呼不同喔！ 如果仅是打包而已，就是“ tar -cv -f file.tar ”而已，这个文件我们称呼为 tarfile 。 如果还有进行压缩的支持，例如“ tar -jcv -f file.tar.bz2 ”时，我们就称呼为 tarball （tar 球？）！这只是一个基本的称谓而已，不过很多书籍与网络都会使用到这个 tarball 的名称！所以得要跟您介绍介绍。

此外，tar 除了可以将数据打包成为文件之外，还能够将文件打包到某些特别的设备去，举例来说， 磁带机 （tape） 就是一个常见的例子。磁带机由于是一次性读取/写入的设备，因此我们不能够使用类似 cp 等指令来复制的！ 那如果想要将 /home, /root, /etc 备份到磁带机（/dev/st0） 时，就可以使用：“tar -cv -f /dev/st0 /home /root /etc”，很简单容易吧！ 磁带机用在备份 （尤其是企业应用） 是很常见的工作喔！

- 特殊应用：利用管线命令与数据流

在 tar 的使用中，有一种方式最特殊，那就是通过标准输入输出的数据流重导向（standard input/standard output）， 以及管线命令 （pipe） 的方式，将待处理的文件一边打包一边解压缩到目标目录去。 关于数据流重导向与管线命令更详细的数据我们会在第十章 bash 再跟

大家介绍， 下面先来看一个例子吧！

```
# 1\. 将 /etc 整个目录一边打包一边在 /tmp 解开
[root@study ~]# cd /tmp
[root@study tmp]# tar -cvf - /etc &#124; tar -xvf -
# 这个动作有点像是 cp -r /etc /tmp 啦～依旧是有其有用途的！
# 要注意的地方在于输出文件变成 - 而输入文件也变成 - ，又有一个 &#124; 存在～
# 这分别代表 standard output, standard input 与管线命令啦！
# 简单的想法中，你可以将 - 想成是在内存中的一个设备（缓冲区）。
# 更详细的数据流与管线命令，请翻到 [bash](../Text/index.html) 章节啰！
```

在上面的例子中，我们想要“将 /etc 下面的数据直接 copy 到目前所在的路径，也就是 /tmp 下面”，但是又觉得使用 cp -r 有点麻烦，那么就直接以这个打包的方式来打包，其中，指令里面的 - 就是表示那个被打包的文件啦！ 由于我们不想要让中间文件存在，所以就以这一个方式来进行复制的行为啦！

- 例题：系统备份范例

系统上有非常多的重要目录需要进行备份，而且其实我们也不建议你将备份数据放置到 /root 目录下！ 假设目前你已经知道重要的目录有下面这几个：

- /etc/ （配置文件）
- /home/ （使用者的主文件夹）
- /var/spool/mail/ （系统中，所有帐号的邮件信箱）
- /var/spool/cron/ （所有帐号的工作排成配置文件）
- /root （系统管理员的主文件夹）

然后我们也知道，由于第七章曾经做过的练习的关系， /home/loop* 不需要备份，而且 /root 下面的压缩文件也不需要备份，另外假设你要将备份的数据放置到 /backups ，并且该目录仅有 root 有权限进入！ 此外，每次备份的文件名都希望不相同，例如使用：backup-system-20150701.tar.bz2 之类的文件名来处理。 那你该如何处理这个备份数据呢？（请先动手作看看，再来察看一下下面的参考解答！）

```
# 1\. 先处理要放置备份数据的目录与权限：
[root@study ~]# mkdir /backups
[root@study ~]# chmod 700 /backups
[root@study ~]# ll -d /backups
drwx------. 2 root root 6 Jul  1 17:25 /backups

# 2\. 假设今天是 2015/07/01 ，则创建备份的方式如下：
[root@study ~]# tar -jcv -f /backups/backup-system-20150701.tar.bz2 \
&gt; --exclude=/root/*.bz2 --exclude=/root/*.gz --exclude=/home/loop* \
&gt; /etc /home /var/spool/mail /var/spool/cron /root
....（过程省略）....

[root@study ~]# ll -h /backups/
-rw-r--r--. 1 root root 21M Jul  1 17:26 backup-system-20150701.tar.bz2
```

- 解压缩后的 SELinux 课题

如果，鸟哥是说如果，如果因为某些缘故，所以你的系统必须要以备份的数据来回填到原本的系统中，那么得要特别注意复原后的系统的 SELinux 问题！ 尤其是在系统文件上面！例如 /etc 下面的文件群。SELinux 是比较特别的细部权限设置，相关的介绍我们会在 16 章好好的介绍一下。 在这里，你只要先知道，SELinux 的权限问题“可能会让你的系统无法存取某些配置文件内容，导致影响到系统的正常使用权”。

这两天 （2015/07） 接到一个网友的 email，他说他使用鸟哥介绍的方法通过 tar 去备份了 /etc 的数据，然后尝试在另一部系统上面复原回来。 复原倒是没问题，但是复原完毕之后，无论如何就是无法正常的登陆系统！明明使用单人维护模式去操作系统时，看起来一切正常～但就是无法顺利登陆。 其实这个问题倒是很常见！大部分原因就是因为 /etc/shadow 这个密码文件的 SELinux 类型在还原时被更改了！导致系统的登陆程序无法顺利的存取它， 才造成无法登陆的窘境。

那如何处理呢？简单的处理方式有这几个：

- 通过各种可行的救援方式登陆系统，然后修改 /etc/selinux/config 文件，将 SELinux 改成 permissive 模式，重新开机后系统就正常了；
- 在第一次复原系统后，不要立即重新开机！先使用 restorecon -Rv /etc 自动修复一下 SELinux 的类型即可。
- 通过各种可行的方式登陆系统，创建 /.autorelabel 文件，重新开机后系统会自动修复 SELinux 的类型，并且又会再次重新开机，之后就正常了！

鸟哥个人是比较偏好第 2 个方法，不过如果忘记了该步骤就重新开机呢？那鸟哥比较偏向使用第 3 个方案来处理，这样就能够解决复原后的 SELinux 问题啰！ 至于更详细的 SELinux ，我们得要讲完程序 （process） 之后，你才会有比较清楚的认知，因此还请慢慢学习，到第 16 章你就知道问题点了！ ^_^

8.4 XFS 文件系统的备份与还原

使用 tar 通常是针对目录树系统来进行备份的工作，那么如果想要针对整个文件系统来进行备份与还原呢？由于 CentOS 7 已经使用 XFS 文件系统作为默认值，所以那个好用的 xfsdump 与 xfsrestore 两个工具对 CentOS 7 来说，就是挺重要的工具软件了。下面就让我们来谈一谈这个指令的用法吧！

8.4.1 XFS 文件系统备份 xfsdump

其实 xfsdump 的功能颇强！他除了可以进行文件系统的完整备份 （full backup） 之外，还可以进行累积备份 （Incremental backup） 喔！ 啥是累积备份呢？这么说好了，假设你的 /home 是独立的一个文件系统，那你在第一次使用 xfsdump 进行完整备份后，等过一段时间的文件系统自然运行后， 你再进行第二次 xfsdump 时，就可以选择累积备份了！此时新备份的数据只会记录与第一次完整备份所有差异的文件而已。 看不懂吗？没关系！我们用一张简图来说明。

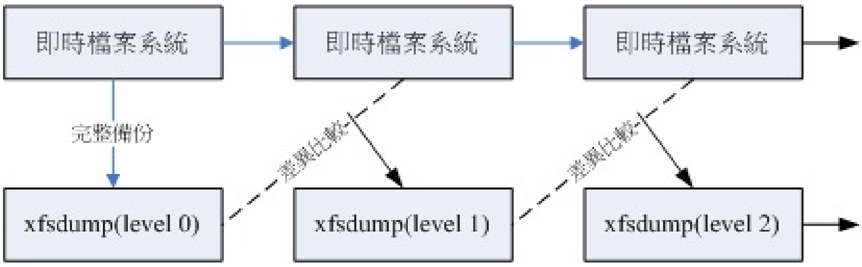


图8.4.1、xfsdump 运行时，完整备份与累积备份示意图

如上图所示，上方的“实时文件系统”是一直随着时间而变化的数据，例如在 /home 里面的文件数据会一直变化一样。 而下面的方块则是 xfsdump 备份起来的数据，第一次备份一定是完整备份，完整备份在 xfsdump 当中被定义为 level 0 喔！等到第二次备份时，/home 文件系统内的数据已经与 level 0 不一样了，而 level 1 仅只是比较目前的文件系统与 level 0 之间的差异后，备份有变化过的文件而已。至于 level 2 则是与 level 1 进行比较啦！这样了解呼？至于各个 level 的纪录档则放置于 /var/lib/xfsdump/inventory 中。

另外，使用 xfsdump 时，请注意下面的限制喔：

- xfsdump 不支持没有挂载的文件系统备份！所以只能备份已挂载的！
- xfsdump 必须使用 root 的权限才能操作 （涉及文件系统的关系）
- xfsdump 只能备份 XFS 文件系统啊！
- xfsdump 备份下来的数据 （文件或储存媒体） 只能让 xfsrestore 解析
- xfsdump 是通过文件系统的 UUID 来分辨各个备份文件的，因此不能备份两个具有相同 UUID 的文件系统喔！

xfsdump 的选项虽然非常的繁复，不过如果只是想要简单的操作时，您只要记得下面的几个选项就很够用了！

```
[root@study ~]# xfsdump [-L S_label] [-M M_label] [-l #] [-f 备份文件] 待备份数据
[root@study ~]# xfsdump -I
选项与参数：
-L  ：xfsdump 会纪录每次备份的 session 标头，这里可以填写针对此文件系统的简易说明
-M  ：xfsdump 可以纪录储存媒体的标头，这里可以填写此媒体的简易说明
-l  ：是 L 的小写，就是指定等级～有 0~9 共 10 个等级喔！ （默认为 0，即完整备份）
-f  ：有点类似 tar 啦！后面接产生的文件，亦可接例如 /dev/st0 设备文件名或其他一般文件文件名等
-I  ：从 /var/lib/xfsdump/inventory 列出目前备份的信息状态
```

特别注意， xfsdump 默认仅支持文件系统的备份，并不支持特定目录的备份～所以你不能用 xfsdump 去备份 /etc ！ 因为 /etc 从来就不是一个独立的文件系统！注意！注意！

- 用 xfsdump 备份完整的文件系统

现在就让我们来做几个范例吧！假设你跟鸟哥一样有将 /boot 分区出自己的文件系统，要整个文件系统备份可以这样作：

```
# 1\. 先确定 /boot 是独立的文件系统喔！
[root@study ~]# df -h /boot
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda2      1014M  131M  884M  13% /boot      # 挂载 /boot 的是 /dev/vda 设备！
# 看！确实是独立的文件系统喔！ /boot 是挂载点！

# 2\. 将完整备份的文件名记录成为 /srv/boot.dump ：
[root@study ~]# xfsdump -l 0 -L boot_all -M boot_all -f /srv/boot.dump /boot
xfsdump -l 0 -L boot_all -M boot_all -f /srv/boot.dump /boot
xfsdump: using file dump （drive_simple） strategy
xfsdump: version 3.1.4 （dump format 3.0） - type ^C for status and control
xfsdump: level 0 dump of study.centos.vbird:/boot              # 开始备份本机/boot系统
xfsdump: dump date: Wed Jul  1 18:43:04 2015                   # 备份的时间
xfsdump: session id: 418b563f-26fa-4c9b-98b7-6f57ea0163b1      # 这次dump的ID
xfsdump: session label: "boot_all"                             # 简单给予一个名字记忆
xfsdump: ino map phase 1: constructing initial dump list       # 开始备份程序
xfsdump: ino map phase 2: skipping （no pruning necessary）
xfsdump: ino map phase 3: skipping （only one dump stream）
xfsdump: ino map construction complete
xfsdump: estimated dump size: 103188992 Bytes
xfsdump: creating dump session media file 0 （media 0, file 0）
xfsdump: dumping ino map
xfsdump: dumping directories
xfsdump: dumping non-directory files
xfsdump: ending media file
xfsdump: media file size 102872168 Bytes
xfsdump: dump size （non-dir files） : 102637296 Bytes
xfsdump: dump complete: 1 seconds elapsed
xfsdump: Dump Summary:
xfsdump:   stream 0 /srv/boot.dump OK （success）
xfsdump: Dump Status: SUCCESS
# 在指令的下达方面，你也可以不加 -L 及 -M 的，只是那就会进入互动模式，要求你 enter！
# 而执行 xfsdump 的过程中会出现如上的一些讯息，您可以自行仔细的观察！

[root@study ~]# ll /srv/boot.dump
-rw-r--r--. 1 root root 102872168 Jul  1 18:43 /srv/boot.dump

[root@study ~]# ll /var/lib/xfsdump/inventory
-rw-r--r--. 1 root root 5080 Jul  1 18:43 506425d2-396a-433d-9968-9b200db0c17c.StObj
-rw-r--r--. 1 root root  312 Jul  1 18:43 94ac5f77-cb8a-495e-a65b-2ef7442b837c.InvIndex
-rw-r--r--. 1 root root  576 Jul  1 18:43 fstab
# 使用了 xfsdump 之后才会有上述 /var/lib/xfsdump/inventory 内的文件产生喔！
```

这样很简单的就创建起来 /srv/boot.dump 文件，该文件将整个 /boot/ 文件系统都备份下来了！ 并且将备份的相关信息 （文件系统/时间/session ID等等） 写入 /var/lib/xfsdump/inventory 中，准备让下次备份时可以作为一个参考依据。 现在让我们来进行一个测试，检查看看能否眞的创建 level 1 的备份呢？

- 用 xfsdump 进行累积备份 （Incremental backups）

你一定得要进行过完整备份后 （-l 0） 才能够继续有其他累积备份 （-l 1~9） 的能耐！所以，请确定上面的实做已经完成！ 接下来让我们来搞一搞累积备份功能吧！

```
# 0\. 看一下有没有任何文件系统被 xfsdump 过的数据？
[root@study ~]# xfsdump -I
file system 0:
    fs id:          94ac5f77-cb8a-495e-a65b-2ef7442b837c
    session 0:
        mount point:    study.centos.vbird:/boot
        device:         study.centos.vbird:/dev/vda2
        time:           Wed Jul  1 18:43:04 2015
        session label:  "boot_all"
        session id:     418b563f-26fa-4c9b-98b7-6f57ea0163b1
        level:          0
        resumed:        NO
        subtree:        NO
        streams:        1
        stream 0:
                pathname:       /srv/boot.dump
                start:          ino 132 offset 0
                end:            ino 2138243 offset 0
                interrupted:    NO
                media files:    1
                media file 0:
                        mfile index:    0
                        mfile type:     data
                        mfile size:     102872168
                        mfile start:    ino 132 offset 0
                        mfile end:      ino 2138243 offset 0
                        media label:    "boot_all"
                        media id:       a6168ea6-1ca8-44c1-8d88-95c863202eab
xfsdump: Dump Status: SUCCESS
# 我们可以看到目前仅有一个 session 0 的备份数据而已！而且是 level 0 喔！

# 1\. 先恶搞一下，创建一个大约 10 MB 的文件在 /boot 内：
[root@study ~]# dd if=/dev/zero of=/boot/testing.img bs=1M count=10
10+0 records in
10+0 records out
10485760 Bytes （10 MB） copied, 0.166128 seconds, 63.1 MB/s

# 2\. 开始创建差异备份文件，此时我们使用 level 1 吧：
[root@study ~]# xfsdump -l 1 -L boot_2 -M boot_2 -f /srv/boot.dump1 /boot
....（中间省略）....

[root@study ~]# ll /srv/boot*
-rw-r--r--. 1 root root 102872168 Jul  1 18:43 /srv/boot.dump
-rw-r--r--. 1 root root  10510952 Jul  1 18:46 /srv/boot.dump1
# 看看文件大小，岂不是就是刚刚我们所创建的那个大文件的容量吗？ ^_^

# 3\. 最后再看一下是否有记录 level 1 备份的时间点呢？
[root@study ~]# xfsdump -I
file system 0:
    fs id:          94ac5f77-cb8a-495e-a65b-2ef7442b837c
    session 0:
        mount point:    study.centos.vbird:/boot
        device:         study.centos.vbird:/dev/vda2
....（中间省略）....

    session 1:
        mount point:    study.centos.vbird:/boot
        device:         study.centos.vbird:/dev/vda2
        time:           Wed Jul  1 18:46:21 2015
        session label:  "boot_2"
        session id:     c71d1d41-b3bb-48ee-bed6-d77c939c5ee8
        level:          1
        resumed:        NO
        subtree:        NO
        streams:        1
        stream 0:
                pathname:       /srv/boot.dump1
                start:          ino 455518 offset 0
....（下面省略）....
```

通过这个简单的方式，我们就能够仅备份差异文件的部分啰！

8.4.2 XFS 文件系统还原 xfsrestore

备份文件就是在急用时可以回复系统的重要数据，所以有备份当然就得要学学如何复原了！xfsdump 的复原使用的是 xfsrestore 这个指令！这个指令的选项也非常的多～您可以自行 man xfsrestore 瞧瞧！ 鸟哥在这里仅作个简单的介绍啰！

```
[root@study ~]# xfsrestore -I &lt;==用来察看备份文件数据
[root@study ~]# xfsrestore [-f 备份文件] [-L S_label] [-s] 待复原目录 &lt;==单一文件全系统复原
[root@study ~]# xfsrestore [-f 备份文件] -r 待复原目录 &lt;==通过累积备份文件来复原系统
[root@study ~]# xfsrestore [-f 备份文件] -i 待复原目录 &lt;==进入互动模式
选项与参数：
-I  ：跟 xfsdump 相同的输出！可查询备份数据，包括 Label 名称与备份时间等
-f  ：后面接的就是备份文件！企业界很有可能会接 /dev/st0 等磁带机！我们这里接文件名！
-L  ：就是 Session 的 Label name 喔！可用 -I 查询到的数据，在这个选项后输入！
-s  ：需要接某特定目录，亦即仅复原某一个文件或目录之意！
-r  ：如果是用文件来储存备份数据，那这个就不需要使用。如果是一个磁带内有多个文件，
      需要这东西来达成累积复原
-i  ：进入互动模式，进阶管理员使用的！一般我们不太需要操作它！
```

- 用 xfsrestore 观察 xfsdump 后的备份数据内容

要找出 xfsdump 的内容就使用 xfsrestore -I 来查阅即可！不需要加任何参数！因为 xfsdump 与 xfsrestore 都会到 /var/lib/xfsdump/inventory/ 里面去捞数据来显示的！因此两者输出是相同的！

```
[root@study ~]# xfsrestore -I
file system 0:
    fs id:          94ac5f77-cb8a-495e-a65b-2ef7442b837c
    session 0:
        mount point:    study.centos.vbird:/boot
        device:         study.centos.vbird:/dev/vda2
        time:           Wed Jul  1 18:43:04 2015
        session label:  "boot_all"
        session id:     418b563f-26fa-4c9b-98b7-6f57ea0163b1
        level:          0
                pathname:       /srv/boot.dump
                        mfile size:     102872168
                        media label:    "boot_all"
    session 1:
        mount point:    study.centos.vbird:/boot
        device:         study.centos.vbird:/dev/vda2
        time:           Wed Jul  1 18:46:21 2015
        session label:  "boot_2"
        session id:     c71d1d41-b3bb-48ee-bed6-d77c939c5ee8
        level:          1
                pathname:       /srv/boot.dump1
                        mfile size:     10510952
                        media label:    "boot_2"
xfsrestore: Restore Status: SUCCESS
# 鸟哥已经将不重要的项目删除了，所以上面的输出是经过经简化的结果！
# 我们可以看到这个文件系统是 /boot 载点，然后有两个备份，一个 level 0 一个 level 1。
# 也看到这两个备份的数据他的内容大小！更重要的，就是那个 session label 喔！
```

这个查询重点是找出到底哪个文件是哪个挂载点？而该备份文件又是什么 level 等等的！接下来，让我们实做一下从备份还原系统吧！

- 简单复原 level 0 的文件系统

先来处理一个简单的任务，就是将 /boot 整个复原到最原本的状态～你该如何处理？其实很简单，我们只要知道想要被复原的那个文件， 以及该文件的 session label name，就可以复原啦！我们从上面的观察已经知道 level 0 的 session label 是“boot_all”啰！那整个流程是这样：

```
# 1\. 直接将数据给它覆盖回去即可！
[root@study ~]# xfsrestore -f /srv/boot.dump -L boot_all /boot
xfsrestore: using file dump （drive_simple） strategy
xfsrestore: version 3.1.4 （dump format 3.0） - type ^C for status and control
xfsrestore: using online session inventory
xfsrestore: searching media for directory dump
xfsrestore: examining media file 0
xfsrestore: reading directories
xfsrestore: 8 directories and 327 entries processed
xfsrestore: directory post-processing
xfsrestore: restoring non-directory files
xfsrestore: restore complete: 1 seconds elapsed
xfsrestore: Restore Summary:
xfsrestore:   stream 0 /srv/boot.dump OK （success）  # 是否是正确的文件啊？
xfsrestore: Restore Status: SUCCESS

# 2\. 将备份数据在 /tmp/boot 下面解开！
[root@study ~]# mkdir /tmp/boot
[root@study ~]# xfsrestore -f /srv/boot.dump -L boot_all /tmp/boot
[root@study ~]# du -sm /boot /tmp/boot
109     /boot
99      /tmp/boot
# 咦！两者怎么大小不一致呢？没关系！我们来检查看看！

[root@study ~]# diff -r /boot /tmp/boot
Only in /boot: testing.img
# 看吧！原来是 /boot 我们有增加过一个文件啦！
```

因为原本 /boot 里面的东西我们没有删除，直接复原的结果就是：“同名的文件会被覆盖，其他系统内新的文件会被保留”喔！所以， 那个 /boot/testing.img 就会一直在里头～如果备份的目的地是新的位置，当然就只有原本备份的数据而已啊！那个 diff -r 可以比较两个目录内的文件差异！ 通过该指令我们可以找到两个目录的差异处！

```
# 3\. 仅复原备份文件内的 grub2 到 /tmp/boot2/ 里头去！
[root@study ~]# mkdir /tmp/boot2
[root@study ~]# xfsrestore -f /srv/boot.dump -L boot_all -s grub2 /tmp/boot2
```

如果只想要复原某一个目录或文件的话，直接加上“ -s 目录 ”这个选项与参数即可！相当简单好用！

- 复原累积备份数据

其实复原累积备份与复原单一文件系统相似耶！如果备份数据是由 level 0 -> level 1 -> level 2... 去进行的， 当然复原就得要相同的流程来复原！因此当我们复原了 level 0 之后，接下来当然就要复原 level 1 到系统内啊！ 我们可以前一个案例复原 /tmp/boot 的情况来继续往下处

理：

```
# 继续复原 level 1 到 /tmp/boot 当中！
[root@study ~]# xfsrestore -f /srv/boot.dump1 /tmp/boot
```

- 仅还原部分文件的 xfsrestore 互动模式

刚刚的 -s 可以接部份数据来还原，但是...如果我就根本不知道备份文件里面有啥文件，那该如何选择啊？用猜的喔？ 又如果要复原的文件数量太多时，用 -s 似乎也是笨笨的～那怎办？有没有比较好的方式呢？有的，就通过 -i 这个互动界面吧！ 举例来说，我们想要知道 level 0 的备份数据里面有哪些东西，然后再少量的还原回来的话！

```
# 1\. 先进入备份文件内，准备找出需要备份的文件名数据，同时预计还原到 /tmp/boot3 当中！
[root@study ~]# mkdir /tmp/boot3
[root@study ~]# xfsrestore -f /srv/boot.dump -i /tmp/boot3
 ========================== subtree selection dialog ==========================

the following commands are available:
        pwd
        ls [ &lt;path&gt; ]
        cd [ &lt;path&gt; ]
        add [ &lt;path&gt; ]       # 可以加入复原文件列表中
        delete [ &lt;path&gt; ]    # 从复原列表拿掉文件名！并非删除喔！
        extract              # 开始复原动作！
        quit
        help

 -&gt; ls
          455517 initramfs-3.10.0-229.el7.x86_64kdump.img
             138 initramfs-3.10.0-229.el7.x86_64.img
             141 initrd-plymouth.img
             140 vmlinuz-0-rescue-309eb890d09f440681f596543d95ec7a
             139 initramfs-0-rescue-309eb890d09f440681f596543d95ec7a.img
             137 vmlinuz-3.10.0-229.el7.x86_64
             136 symvers-3.10.0-229.el7.x86_64.gz
             135 config-3.10.0-229.el7.x86_64
             134 System.map-3.10.0-229.el7.x86_64
             133 .vmlinuz-3.10.0-229.el7.x86_64.hmac
         1048704 grub2/
             131 grub/

 -&gt; add grub
 -&gt; add grub2
 -&gt; add config-3.10.0-229.el7.x86_64
 -&gt; extract

[root@study ~]# ls -l /tmp/boot3
-rw-r--r--. 1 root root 123838 Mar  6 19:45 config-3.10.0-229.el7.x86_64
drwxr-xr-x. 2 root root     26 May  4 17:52 grub
drwxr-xr-x. 6 root root    104 Jun 25 00:02 grub2
# 就只会有 3 个文件名被复原，当然，如果文件名是目录，那下面的子文件当然也会被还原回来的！
```

事实上，这个 -i 是很有帮助的一个项目！可以从备份文件里面找出你所需要的数据来复原！相当有趣！当然啦， 如果你已经知道文件名，使用 -s 不需要进入备份文件就能够处理掉这部份了！

8.5 光盘写入工具

事实上，企业还是挺爱用磁带来进行备份的，容量高、储存时限长、挺耐摔等等，至于以前很热门的 DVD/CD 等，则因为储存速度慢、 容量没有大幅度提升，所以目前除了行政部门为了“归档”而需要的工作之外，这个咚咚的存在性已经被 U盘所取代了。 你可能会谈到说，不是还有蓝光嘛？但这家伙目前主要应用还是在多媒体影音方面，如果要大容量的储存，个人建议，还是使用 USB 外接式硬盘， 一颗好几个 TB 给你用，不是更爽嘛？所以，鸟哥是认为，DVD/CD 虽然还是有存在的价值 （例如前面讲的归档），不过，越来越少人使用了。

虽然很少使用，不过，某些特别的情况下，没有这东西又不行～因此，我们还是来介绍一下创建光盘镜像文件以及烧录软件吧！ 否则，偶而需要用到时，找不到软件数据还挺伤脑筋的！文字模式的烧录行为要怎么处理呢？通常的作法是这样的：

- 先将所需要备份的数据创建成为一个镜像文件（iso），利用 mkisofs 指令来处理；
- 将该镜像文件烧录至光盘或 DVD 当中，利用 cdrecord 指令来处理。

下面我们就分别来谈谈这两个指令的用法吧！

8.5.1 mkisofs：创建镜像文件

烧录可开机与不可开机的光盘，使用的方法不太一样喔 ！

- 制作一般数据光盘镜像文件

我们从 FTP 站捉下来的 Linux 镜像文件 （不管是 CD 还是 DVD） 都得要继续烧录成为实体的光盘/DVD 后， 才能够进一步的使用，包括安装或更新你的 Linux 啦！同样的道理，你想要利用烧录机将你的数据烧录到 DVD 时， 也得要先将你的数据包成一个镜像文件，这样才能够写入DVD片中。而将你的数据包成一个镜像文件的方式就通过 mkisofs 这个指令即可。 mkisofs 的使用方式如下：

```
[root@study ~]# mkisofs [-o 镜像文件] [-Jrv] [-V vol] [-m file] 待备份文件... \
&gt;  -graft-point isodir=systemdir ...
选项与参数：
-o ：后面接你想要产生的那个镜像文件文件名。
-J ：产生较相容于 windows 机器的文件名结构，可增加文件名长度到 64 个 unicode 字符
-r ：通过 Rock Ridge 产生支持 Unix/Linux 的文件数据，可记录较多的信息（如 UID/GID等） ；
-v ：显示创建 ISO文件的过程
-V vol  ：创建 Volume，有点像 Windows 在文件资源管理器内看到的 CD title 的东西
-m file ：-m 为排除文件 （exclude） 的意思，后面的文件不备份到镜像文件中，也能使用 * 万用字符喔
-graft-point：graft有转嫁或移植的意思，相关数据在下面文章内说明。
```

其实 mkisofs 有非常多好用的选项可以选择，不过如果我们只是想要制作“数据光盘”时，上述的选项也就够用了。 光盘的格式一般称为 iso9660 ，这种格式一般仅支持旧版的 DOS 文件名，亦即文件名只能以 8.3 （文件名8个字符，扩展名3个字符） 的方式存在。如果加上 -r 的

选项之后，那么文件信息能够被记录的比较完整，可包括UID/GID与权限等等！ 所以，记得加这个 -r 的选项。

此外，一般默认的情况下，所有要被加到镜像文件中的文件都会被放置到镜像文件中的根目录， 如此一来可能会造成烧录后的文件分类不易的情况。所以，你可以使用 -graft-point 这个选项，当你使用这个选项之后， 可以利用如下的方法来定义位于镜像文件中的目录，例如：

- 镜像文件中的目录所在=实际 Linux 文件系统的目录所在
- /movies/=/srv/movies/ （在 Linux 的 /srv/movies 内的文件，加至镜像文件中的 /movies/ 目录）
- /linux/etc=/etc （将 Linux 中的 /etc/ 内的所有数据备份到镜像文件中的 /linux/etc/ 目录中）

我们通过一个简单的范例来说明一下吧。如果你想要将 /root, /home, /etc 等目录内的数据通通烧录起来的话， 先得要处理一下镜像文件，我们先不使用 -graft-point 的选项来处理这个镜像文件试看看：

```
[root@study ~]# mkisofs -r -v -o /tmp/system.img /root /home /etc
I: -input-charset not specified, using utf-8 (detected in locale settings)
genisoimage 1.1.11 (Linux)
Scanning /root
.....(中间省略).....
Scanning /etc/scl/prefixes
Using SYSTE000.;1 for  /system-release-cpe (system-release)      # 被改名子了！
Using CENTO000.;1 for  /centos-release-upstream (centos-release) # 被改名子了！
Using CRONT000.;1 for  /crontab (crontab)
genisoimage: Error: '/etc/crontab' and '/root/crontab' have the same Rock Ridge name 'cro
Unable to sort directory                                        # 文件名不可一样啊！
NOTE: multiple source directories have been specified and merged into the root
of the filesystem. Check your program arguments. genisoimage is not tar.
# 看到没？因为文件名一模一样，所以就不给你创建 ISO 档了啦！
# 请先删除 /root/crontab 这个文件，然后再重复执行一次 mkisofs 吧！

[root@study ~]# rm /root/crontab
[root@study ~]# mkisofs -r -v -o /tmp/system.img /root /home /etc
.....(前面省略).....
 83.91% done, estimate finish Thu Jul  2 18:48:04 2015
 92.29% done, estimate finish Thu Jul  2 18:48:04 2015
Total translation table size: 0
Total rockridge attributes Bytes: 600251
Total directory Bytes: 2150400
Path table size(Bytes): 12598
Done with: The File(s)                       Block(s)     58329
Writing:   Ending Padblock                  Start Block 59449
Done with: Ending Padblock                  Block(s)     150
Max brk space used 548000
59599 extents written (116 MB)

[root@study ~]# ll -h /tmp/system.img
-rw-r--r--. 1 root root 117M Jul  2 18:48 /tmp/system.img

[root@study ~]# mount -o loop /tmp/system.img /mnt
[root@study ~]# df -h /mnt
Filesystem      Size  Used Avail Use% Mounted on
/dev/loop0      117M  117M     0 100% /mnt

[root@study ~]# ls /mnt
abrt            festival          mail.rc                   rsyncd.conf
adjtime         filesystems       makedumpfile.conf.sample  rsyslog.conf
alex            firewalld         man_db.conf               rsyslog.d
# 看吧！一堆数据都放置在一起！包括有的没有的目录与文件等等！

[root@study ~]# umount /mnt
# 测试完毕要记得卸载！
```

由上面的范例我们可以看到，三个目录（/root, /home, /etc）的数据通通放置到了镜像文件的最顶层目录中！ 眞是不方便～尤其由于 /root/etc 的存在，导致那个 /etc 的数据似乎没有被包含进来的样子！眞不合理～ 此时我们可以使用 -graft-point 来处理啰！

```
[root@study ~]# mkisofs -r -V 'linux_file' -o /tmp/system.img \
&gt;  -m /root/etc -graft-point /root=/root /home=/home /etc=/etc
[root@study ~]# ll -h /tmp/system.img
-rw-r--r--. 1 root root 92M Jul  2 19:00 /tmp/system.img
# 上面的指令会创建一个大文件，其中 -graft-point 后面接的就是我们要备份的数据。
# 必须要注意的是那个等号的两边，等号左边是在镜像文件内的目录，右侧则是实际的数据。

[root@study ~]# mount -o loop /tmp/system.img /mnt
[root@study ~]# ll /mnt
dr-xr-xr-x. 131 root root 34816 Jun 26 22:14 etc
dr-xr-xr-x.   5 root root  2048 Jun 17 00:20 home
dr-xr-xr-x.   8 root root  4096 Jul  2 18:48 root
# 瞧！数据是分门别类的在各个目录中喔这样了解乎？最后将数据卸载一下：

[root@study ~]# umount /mnt
```

如果你想要将实际的数据直接倒进 ISO 档中，那就得要使用这个 -graft-point 来处理处理比较妥当！不然没有分第一层目录， 后面的数据管理实在是很麻烦。如果你是有自己要制作的数据内容，其实最简单的方法，就是将所有的数据预先处理到某一个目录中， 再烧录该目录即可！例如上述的 /etc, /root, /home 先全部复制到 /srv/cdrom 当中，然后跑到 /srv/cdrom 当中， 再使用类似“ mkisofs -r -v -o /tmp/system.img . ”的方式来处理即可！这样也比较单纯～

- 制作/修改可开机光盘图像挡

在鸟哥的研究室中，学生常被要求要制作“一键安装”的安装光盘！也就是说，得要修改原版的光盘镜像文件，改成可以自动载入某些程序的流程， 让这片光盘放入主机光驱后，只要开机利用光盘片来开机，那就直接安装系统，不再需要询问管理员一些有的没有的！等于是自动化处理啦！ 那些流程比较麻烦，因为得要知道 kickstart 的相关技术等，那个我们先不谈，这里要谈的是，那如何让这片光盘的内容被修改之后， 还可以烧录成为可开机的模样呢？

因为鸟哥这部测试机的容量比较小，又仅是测试而已啊，因此鸟哥选择 CentOS-7-x86_64-Minimal-1503-01.iso 这个最小安装光盘镜像文件来测试给各位瞧瞧！ 假设你已经到昆山科大 http://ftp.ksu.edu.tw/FTP/CentOS/7/isos/x86_64/ 取得了最小安装的 Image 档，而且放在 /home 下面～之后我们要将里头的数据进行修改，假设新的镜像文件目录放置于 /srv/newcd 里面，那你应该要这样做：

```
# 1\. 先观察一下这片光盘里面有啥东西？是否是我们需要的光盘系统！
[root@study ~]# isoinfo -d -i /home/CentOS-7-x86_64-Minimal-1503-01.iso
CD-ROM is in ISO 9660 format
System id: LINUX
Volume id: CentOS 7 x86_64
Volume set id:
Publisher id:
Data preparer id:
Application id: GENISOIMAGE ISO 9660/HFS FILESYSTEM CREATOR （C） 1993 E.YOUNGDALE （C） .
Copyright File id:
.....（中间省略）.....
    Eltorito defaultboot header:
 Bootid 88 （bootable）
        Boot media 0 （No Emulation Boot）
        Load segment 0
        Sys type 0
        Nsect 4

# 2\. 开始挂载这片光盘到 /mnt ，并且将所有数据完整复制到 /srv/newcd 目录去喔
[root@study ~]# mount /home/CentOS-7-x86_64-Minimal-1503-01.iso /mnt
[root@study ~]# mkdir /srv/newcd
[root@study ~]# rsync -a /mnt/ /srv/newcd
[root@study ~]# ll /srv/newcd/
-rw-r--r--. 1 root root    16 Apr  1 07:11 CentOS_BuildTag
drwxr-xr-x. 3 root root    33 Mar 28 06:34 EFI
-rw-r--r--. 1 root root   215 Mar 28 06:36 EULA
-rw-r--r--. 1 root root 18009 Mar 28 06:36 GPL
drwxr-xr-x. 3 root root    54 Mar 28 06:34 images
drwxr-xr-x. 2 root root  4096 Mar 28 06:34 isolinux
drwxr-xr-x. 2 root root    41 Mar 28 06:34 LiveOS
drwxr-xr-x. 2 root root 20480 Apr  1 07:11 Packages
drwxr-xr-x. 2 root root  4096 Apr  1 07:11 repodata
-rw-r--r--. 1 root root  1690 Mar 28 06:36 RPM-GPG-KEY-CentOS-7
-rw-r--r--. 1 root root  1690 Mar 28 06:36 RPM-GPG-KEY-CentOS-Testing-7
-r--r--r--. 1 root root  2883 Apr  1 07:15 TRANS.TBL
# rsync 可以完整的复制所有的权限属性等数据，也能够进行镜像处理！相当好用的指令喔！
# 这里先了解一下即可。现在 newcd/ 目录内已经是完整的镜像文件内容！

# 3\. 假设已经处理完毕你在 /srv/newcd 里面所要进行的各项修改行为，准备创建 ISO 档！
[root@study ~]# ll /srv/newcd/isolinux/
-r--r--r--. 1 root root     2048 Apr  1 07:15 boot.cat      # 开机的型号数据等等
-rw-r--r--. 1 root root       84 Mar 28 06:34 boot.msg
-rw-r--r--. 1 root root      281 Mar 28 06:34 grub.conf
-rw-r--r--. 1 root root 35745476 Mar 28 06:31 initrd.img
-rw-r--r--. 1 root root    24576 Mar 28 06:38 isolinux.bin  # 相当于开机管理程序
-rw-r--r--. 1 root root     3032 Mar 28 06:34 isolinux.cfg
-rw-r--r--. 1 root root   176500 Sep 11  2014 memtest
-rw-r--r--. 1 root root      186 Jul  2  2014 splash.png
-r--r--r--. 1 root root     2438 Apr  1 07:15 TRANS.TBL
-rw-r--r--. 1 root root 33997348 Mar 28 06:33 upgrade.img
-rw-r--r--. 1 root root   153104 Mar  6 13:46 vesamenu.c32
-rwxr-xr-x. 1 root root  5029136 Mar  6 19:45 vmlinuz       # Linux 核心文件

[root@study ~]# cd /srv/newcd
[root@study newcd]# mkisofs -o /custom.iso -b isolinux/isolinux.bin -c isolinux/boot.cat
&gt; -no-emul-boot -V 'CentOS 7 x86_64' -boot-load-size 4 -boot-info-table -R -J -v -T .
```

此时你就有一个 /custom.img 的文件存在，可以将该光盘烧录出来啰！就这么简单！

8.5.2 cdrecord：光盘烧录工具

新版的 CentOS 7 使用的是 wodim 这个文字界面指令来进行烧录的行为。不过为了相容于旧版的 cdrecord 这个指令，因此 wodim 也有链接到 cdrecord 就是了！因此，你还是可以使用 cdrecord 这个指令。不过，鸟哥建议还是改用 wodim 比较干脆！这个指令常见的选项有下面数个：

```
[root@study ~]# wodim --devices dev=/dev/sr0...            &lt;==查询烧录机的 BUS 位置
[root@study ~]# wodim -v dev=/dev/sr0 blank=[fast&#124;all]       &lt;==抹除重复读写片
[root@study ~]# wodim -v dev=/dev/sr0 -format               &lt;==格式化DVD+RW
[root@study ~]# wodim -v dev=/dev/sr0 [可用选项功能] file.iso
选项与参数：
--devices        ：用在扫瞄磁盘总线并找出可用的烧录机，后续的设备为 ATA 接口
-v               ：在 cdrecord 运行的过程中，显示过程而已。
dev=/dev/sr0     ：可以找出此光驱的 bus 位址，非常重要！
blank=[fast&#124;all]：blank 为抹除可重复写入的CD/DVD-RW，使用fast较快，all较完整
-format          ：对光盘片进行格式化，但是仅针对 DVD+RW 这种格式的 DVD 而已；
[可用选项功能] 主要是写入 CD/DVD 时可使用的选项，常见的选项包括有：
   -data    ：指定后面的文件以数据格式写入，不是以 CD 音轨(-audio)方式写入！
   speed=X  ：指定烧录速度，例如CD可用 speed=40 为40倍数，DVD则可用 speed=4 之类
   -eject   ：指定烧录完毕后自动退出光盘
   fs=Ym    ：指定多少缓冲内存，可用在将镜像文件先暂存至缓冲内存。默认为 4m，
              一般建议可增加到 8m ，不过，还是得视你的烧录机而定。
针对 DVD 的选项功能：
   driveropts=burnfree ：打开 Buffer Underrun Free 模式的写入功能
   -sao                ：支持 DVD-RW 的格式
```

- 侦测你的烧录机所在位置：

文字模式的烧录确实是比较麻烦的，因为没有所见即所得的环境嘛！要烧录首先就得要找到烧录机才行！而由于早期的烧录机都是使用 SCSI 接口，因此查询烧录机的方法就得要配合着 SCSI 接口的认定来处理了。查询烧录机的方式为：

```
[root@study ~]# ll /dev/sr0
brw-rw----+ 1 root cdrom 11, 0 Jun 26 22:14 /dev/sr0 # 一般 Linux 光驱文件名！

[root@study ~]# wodim --devices dev=/dev/sr0
-------------------------------------------------------------------------
 0  dev='/dev/sr0'      rwrw-- : 'QEMU' 'QEMU DVD-ROM'
-------------------------------------------------------------------------

[root@demo ~]# wodim --devices dev=/dev/sr0
wodim: Overview of accessible drives (1 found) :
-------------------------------------------------------------------------
 0  dev='/dev/sr0'      rwrw-- : 'ASUS' 'DRW-24D1ST'
-------------------------------------------------------------------------
# 你可以发现到其实鸟哥做了两个测试！上面的那部主机系统是虚拟机，当然光驱也是仿真的，没法用。
# 因此在这里与下面的 wodim 用法，鸟哥只能使用另一部 Demo 机器测试给大家看了！
```

因为上面那部机器是虚拟机内的虚拟光驱 (QEMU DVD-ROM)，那个无法塞入真正的光盘片啦！真讨厌～所以鸟哥只好找另一部实体 CentOS 7 的主机系统来测试。因此你可以看到下面那部使用的就是正统的 ASUS 光驱了！这样会查阅了吗？注意喔，一定要有 dev=/dev/xxx 那一段，不然系统会告诉你找不到光盘！这真的是很奇怪！不过，反正我们知道光驱的文件名为 /dev/sr0 之类的，直接带入即可。

- 进行 CD/DVD 的烧录动作：

好了，那么现在要如何将 /tmp/system.img 烧录到 CD/DVD 里面去呢？因为要节省空间与避免浪费，鸟哥拿之前多买的可重复读写的 DVD 四倍数 DVD 片来操作！ 因为是可抹除的 DVD，因此可能得要在烧录前先抹除 DVD 片里面的数据才行喔！

```
# 0\. 先抹除光盘的原始内容：（非可重复读写则可略过此步骤）
[root@demo ~]# wodim -v dev=/dev/sr0 blank=fast
# 中间会跑出一堆讯息告诉你抹除的进度，而且会有 10 秒钟的时间等待你的取消！

# 1\. 开始烧录：
[root@demo ~]# wodim -v dev=/dev/sr0 speed=4 -dummy -eject /tmp/system.img
....（前面省略）....
Waiting for reader process to fill input buffer ... input buffer ready.
Starting new track at sector: 0
Track 01:   86 of   86 MB written （fifo 100%） [buf  97%]   4.0x.       # 这里有流程时间！
Track 01: Total Bytes read/written: 90937344/90937344 （44403 sectors）.
Writing  time:   38.337s                                                # 写入的总时间
Average write speed   1.7x.                                             # 换算下来的写入时间
Min drive buffer fill was 97%
Fixating...
Fixating time:  120.943s
wodim: fifo had 1433 puts and 1433 gets.
wodim: fifo was 0 times empty and 777 times full, min fill was 89%.
# 因为有加上 -eject 这个选项的缘故，因此烧录完成后，DVD 会被退出光驱喔！记得推回去！

# 2\. 烧录完毕后，测试挂载一下，检验内容：
[root@demo ~]# mount /dev/sr0/mnt
[root@demo ~]# df -h /mnt
Filesystem            Size  Used Avail Use% Mounted on
Filesystem      Size  Used Avail Use% Mounted on
/dev/sr0         87M   87M     0 100% /mnt

[root@demo ~]# ll /mnt
dr-xr-xr-x. 135 root root 36864 Jun 30 04:00 etc
dr-xr-xr-x.  19 root root  8192 Jul  2 13:16 root

[root@demo ~]# umount /mnt    &lt;==不要忘了卸载
```

基本上，光盘烧录的指令越来越简单，虽然有很多的参数可以使用，不过，鸟哥认为，学习上面的语法就很足够了！ 一般来说，如果有烧录的需求，大多还是使用图形界面的软件来处理比较妥当～使用文字界面的烧录，真的大部分都是烧录数据光盘较多。 因此，上面的语法已经足够工程师的使用啰！

如果你的 Linux 是用来做为服务器之用的话，那么无时无刻的去想“如何备份重要数据”是相当重要的！ 关于备份我们会在第五篇再仔细的谈一谈，这里你要会使用这些工具即可！

8.6 其他常见的压缩与备份工具

还有一些很好用的工具得要跟大家介绍介绍，尤其是 dd 这个玩意儿呢！

8.6.1 dd

我们在第七章当中的特殊 loop 设备挂载时使用过 dd 这个指令对吧？ 不过，这个指令可不只是制作一个文件而已喔～这个 dd 指令最大的功效，鸟哥认为，应该是在于“备份”啊！ 因为 dd 可以读取磁盘设备的内容（几乎是直接读取扇区"sector"），然后将整个设备备份成一个文件呢！真的是相当的好用啊～ dd 的用途有很多啦～但是我们仅讲一些比较重要的选项，如下：

```
[root@study ~]# dd if="input_file" of="output_file" bs="block_size" count="number"
选项与参数：
if   ：就是 input file 啰～也可以是设备喔！
of   ：就是 output file 喔～也可以是设备；
bs   ：规划的一个 block 的大小，若未指定则默认是 512 Bytes（一个 sector 的大小）
count：多少个 bs 的意思。

范例一：将 /etc/passwd 备份到 /tmp/passwd.back 当中
[root@study ~]# dd if=/etc/passwd of=/tmp/passwd.back
4+1 records in
4+1 records out
2092 Bytes （2.1 kB） copied, 0.000111657 s, 18.7 MB/s
[root@study ~]# ll /etc/passwd /tmp/passwd.back
-rw-r--r--. 1 root root 2092 Jun 17 00:20 /etc/passwd
-rw-r--r--. 1 root root 2092 Jul  2 23:27 /tmp/passwd.back
# 仔细的看一下，我的 /etc/passwd 文件大小为 2092 Bytes，因为我没有设置 bs ，
# 所以默认是 512 Bytes 为一个单位，因此，上面那个 4+1 表示有 4 个完整的 512 Bytes，
# 以及未满 512 Bytes 的另一个 block 的意思啦！事实上，感觉好像是 cp 这个指令啦～

范例二：将刚刚烧录的光驱的内容，再次的备份下来成为图像档
[root@study ~]# dd if=/dev/sr0 of=/tmp/system.iso
177612+0 records in
177612+0 records out
90937344 Bytes （91 MB） copied, 22.111 s, 4.1 MB/s
# 要将数据抓下来用这个方法，如果是要将镜像文件写入 USB 磁盘，就会变如下一个范例啰！

范例三：假设你的 USB 是 /dev/sda 好了，请将刚刚范例二的 image 烧录到 USB 磁盘中
[root@study ~]# lsblk /dev/sda
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
sda    8:0    0   2G  0 disk             # 确实是 disk 而且有 2GB 喔！

[root@study ~]# dd if=/tmp/system.iso of=/dev/sda
[root@study ~]# mount /dev/sda /mnt
[root@study ~]# ll /mnt
dr-xr-xr-x. 131 root root 34816 Jun 26 22:14 etc
dr-xr-xr-x.   5 root root  2048 Jun 17 00:20 home
dr-xr-xr-x.   8 root root  4096 Jul  2 18:48 root
# 如果你不想要使用 DVD 来作为开机媒体，那可以将镜像文件使用这个 dd 写入 USB 磁盘，
# 该磁盘就会变成跟可开机光盘一样的功能！可以让你用 USB 来安装 Linux 喔！速度快很多！

范例四：将你的 /boot 整个文件系统通过 dd 备份下来
[root@study ~]# df -h /boot
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda2      1014M  149M  866M  15% /boot       # 请注意！备份的容量会到 1G 喔！
[root@study ~]# dd if=/dev/vda2 of=/tmp/vda2.img
[root@study ~]# ll -h /tmp/vda2.img
-rw-r--r--. 1 root root 1.0G Jul  2 23:39 /tmp/vda2.img
# 等于是将整个 /dev/vda2 通通捉下来的意思～所以，文件大小会跟整颗磁盘的最大量一样大！
```

其实使用 dd 来备份是莫可奈何的情况，很笨耶！因为默认 dd 是一个一个扇区去读/写的，而且即使没有用到的扇区也会倍写入备份文件中！ 因此这个文件会变得跟原本的磁盘一模一样大！不像使用 xfsdump 只备份文件系统中有使用到的部份。不过， dd 就是因为不理会文件系统， 单纯有啥纪录啥，因此不论该磁盘内的文件系统你是否认识，它都可以备份、还原的！所以，鸟哥认为，上述的第三个案例是比较重要的学习喔！

例题：你想要将你的 /dev/vda2 进行完整的复制到另一个 partition 上，请使用你的系统上面未分区完毕的容量再创建一个与 /dev/vda2 差不多大小的分区 （只能比 /dev/vda2 大，不能比他小！），然后将之进行完整的复制 （包括需要复制 boot sector 的区块）。答：因为我们的 /dev/sda 也是个测试的 USB 磁盘，可以随意恶搞！我们刚刚也才测试过将光盘镜像文件给它复制进去而已。 现在，请你分区 /dev/sda1 出来，然后将 /dev/vda2 完整的拷贝进去 /dev/sda1 吧！

```
# 1\. 先进行分区的动作
[root@study ~]# fdisk /dev/sda

Command （m for help）: n
Partition type:
   p   primary （0 primary, 0 extended, 4 free）
   e   extended
Select （default p）: p
Partition number （1-4, default 1）: 1
First sector （2048-4195455, default 2048）: Enter
Using default value 2048
Last sector, +sectors or +size{K,M,G} （2048-4195455, default 4195455）: Enter
Using default value 4195455
Partition 1 of type Linux and of size 2 GiB is set

Command （m for help）: p
   Device Boot      Start         End      Blocks   Id  System
/dev/sda1            2048     4195455     2096704   83  Linux

Command （m for help）: w

[root@study ~]# partprobe

# 2\. 不需要格式化，直接进行 sector 表面的复制！
[root@study ~]# dd if=/dev/vda2 of=/dev/sda1
2097152+0 records in
2097152+0 records out
1073741824 Bytes （1.1 GB） copied, 71.5395 s, 15.0 MB/s

[root@study ~]# xfs_repair -L /dev/sda1  # 一定要先清除一堆 log 才行！
[root@study ~]# uuidgen                  # 下面两行在给予一个新的 UUID
896c38d1-bcb5-475f-83f1-172ab38c9a0c
[root@study ~]# xfs_admin -U 896c38d1-bcb5-475f-83f1-172ab38c9a0c /dev/sda1
# 因为 XFS 文件系统主要使用 UUID 来分辨文件系统，但我们使用 dd 复制，连 UUID
# 也都复制成为相同！当然就得要使用上述的 xfs_repair 及 xfs_admin 来修订一下！

[root@study ~]# mount /dev/sda1 /mnt
[root@study ~]# df -h /boot /mnt
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda2      1014M  149M  866M  15% /boot
/dev/sda1      1014M  149M  866M  15% /mnt
# 这两个玩意儿会“一模一样”喔！

# 3\. 接下来！让我们将文件系统放大吧！！！
[root@study ~]# xfs_growfs /mnt
[root@study ~]# df -h /boot /mnt
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda2      1014M  149M  866M  15% /boot
/dev/sda1       2.0G  149M  1.9G   8% /mnt

[root@study ~]# umount /mnt
```

非常有趣的范例吧！新分区出来的 partition 不需要经过格式化，因为 dd 可以将原本旧的 partition 上面，将 sector 表面的数据整个复制过来！ 当然连同 superblock, boot sector, meta data 等等通通也会复制过来！是否很有趣呢？未来你想要创建两颗一模一样的磁盘时， 只要下达类似： dd if=/dev/sda of=/dev/sdb ，就能够让两颗磁盘一模一样，甚至 /dev/sdb 不需要分区与格式化， 因为该指令可以将 /dev/sda 内的所有数据，包括 MBR 与 partition table 也复制到 /dev/sdb 说！ ^_^

话说，用 dd 来处理这方面的事情真的是很方便，你也不需考虑到啥有的没的，通通是磁盘表面的复制而已！不过如果真的用在文件系统上面， 例如上面这个案例，那么再次挂载时，恐怕得要理解一下每种文件系统的挂载要求！以上面的案例来说，你就得要先清除 XFS 文件系

统内的 log 之后， 重新给予一个跟原本不一样的 UUID 后，才能够顺利挂载！同时，为了让系统继续利用后续没有用到的磁盘空间，那个 xfs_growfs 就得要理解一下。 关于 xfs_growfs 我们会在后续第十四章继续强调！这里先理解即可。

8.6.2 cpio

这个指令挺有趣的，因为 cpio 可以备份任何东西，包括设备设备文件。不过 cpio 有个大问题， 那就是 cpio 不会主动的去找文件来备份！啊！那怎办？所以啰，一般来说， cpio 得要配合类似 find 等可以找到文件名的指令来告知 cpio 该被备份的数据在哪里啊！ 有点小麻烦啦～因为牵涉到我们在第三篇才会谈到的数据流重导向说～ 所以这里你就先背一下语法，等到第三篇讲完你就知道如何使用 cpio 啰！

```
[root@study ~]# cpio -ovcB  &gt; [file&#124;device] &lt;==备份
[root@study ~]# cpio -ivcdu &lt; [file&#124;device] &lt;==还原
[root@study ~]# cpio -ivct  &lt; [file&#124;device] &lt;==察看
备份会使用到的选项与参数：
  -o ：将数据 copy 输出到文件或设备上
  -B ：让默认的 Blocks 可以增加至 5120 Bytes ，默认是 512 Bytes ！
       这样的好处是可以让大文件的储存速度加快（请参考 i-nodes 的观念）
还原会使用到的选项与参数：
  -i ：将数据自文件或设备 copy 出来系统当中
  -d ：自动创建目录！使用 cpio 所备份的数据内容不见得会在同一层目录中，因此我们
       必须要让 cpio 在还原时可以创建新目录，此时就得要 -d 选项的帮助！
  -u ：自动的将较新的文件覆盖较旧的文件！
  -t ：需配合 -i 选项，可用在"察看"以 cpio 创建的文件或设备的内容
一些可共享的选项与参数：
  -v ：让储存的过程中文件名称可以在屏幕上显示
  -c ：一种较新的 portable format 方式储存
```

你应该会发现一件事情，就是上述的选项与指令中怎么会没有指定需要备份的数据呢？还有那个大于 （>） 与小于 （<） 符号是怎么回事啊？因为 cpio 会将数据整个显示到屏幕上，因此我们可以通过将这些屏幕的数据重新导向 （>） 一个新的文件！ 至于还原呢？就是将备份文件读进来 cpio （<） 进行处理之意！我们来进行几个案例你就知道啥是啥了！

```
范例：找出 /boot 下面的所有文件，然后将他备份到 /tmp/boot.cpio 去！
[root@study ~]# cd /
[root@study /]# find boot -print
boot
boot/grub
boot/grub/splash.xpm.gz
....（以下省略）....
# 通过 find 我们可以找到 boot 下面应该要存在的文件名！包括文件与目录！但请千万不要是绝对路径！

[root@study /]# find boot &#124; cpio -ocvB &gt; /tmp/boot.cpio
[root@study /]# ll -h /tmp/boot.cpio
-rw-r--r--. 1 root root 108M Jul  3 00:05 /tmp/boot.cpio
[root@study ~]# file /tmp/boot.cpio
/tmp/boot.cpio: ASCII cpio archive （SVR4 with no CRC）
```

我们使用 find boot 可以找出文件名，然后通过那条管线 （|, 亦即键盘上的 shift+\ 的组合）， 就能将文件名传给 cpio 来进行处理！最终会得到 /tmp/boot.cpio 那个文件喔！你可能会觉得奇怪，为啥鸟哥要先转换目录到 / 再去找 boot 呢？ 为何不能直接找 /boot 呢？这是因为 cpio

很笨！它不会理会你给的是绝对路径还是相对路径的文件名，所以如果你加上绝对路径的 / 开头， 那么未来解开的时候，它就一定会覆盖掉原本的 /boot 耶！那就太危险了！这个我们在 tar 也稍微讲过那个 -P 的选项！！理解吧！ 好了，那接下来让我们来进行解压缩看看。

```
范例：将刚刚的文件给他在 /root/ 目录下解开
[root@study ~]# cd ~
[root@study ~]# cpio -idvc &lt; /tmp/boot.cpio
[root@study ~]# ll /root/boot
# 你可以自行比较一下 /root/boot 与 /boot 的内容是否一模一样！
```

事实上 cpio 可以将系统的数据完整的备份到磁带机上头去喔！如果你有磁带机的话！

- 备份：find / | cpio -ocvB > /dev/st0
- 还原：cpio -idvc < /dev/st0

这个 cpio 好像不怎么好用呦！但是，他可是备份的时候的一项利器呢！因为他可以备份任何的文件， 包括 /dev 下面的任何设备文件！所以他可是相当重要的呢！而由于 cpio 必需要配合其他的程序，例如 find 来创建文件名，所以 cpio 与管线命令及数据流重导向的相关性就相当的重要了！

其实系统里面已经含有一个使用 cpio 创建的文件喔！那就是 /boot/initramfs-xxx 这个文件啦！现在让我们来将这个文件解压缩看看，看你能不能发现该文件的内容为何？

```
# 1\. 我们先来看看该文件是属于什么文件格式，然后再加以处理：
[root@study ~]# file /boot/initramfs-3.10.0-229.el7.x86_64.img
/boot/initramfs-3.10.0-229.el7.x86_64.img: ASCII cpio archive （SVR4 with no CRC）

[root@study ~]# mkdir /tmp/initramfs
[root@study ~]# cd /tmp/initramfs
[root@study initramfs]# cpio -idvc &lt; /boot/initramfs-3.10.0-229.el7.x86_64.img
.
kernel
kernel/x86
kernel/x86/microcode
kernel/x86/microcode/GenuineIntel.bin
early_cpio
22 blocks
# 瞧！这样就将这个文件解开啰！这样了解乎？
```

8.7 重点回顾

- 压缩指令为通过一些运算方法去将原本的文件进行压缩，以减少文件所占用的磁盘容量。 压缩前与压缩后的文件所占用的磁盘容量比值， 就可以被称为是“压缩比”
- 压缩的好处是可以减少磁盘容量的浪费，在 WWW 网站也可以利用文件压缩的技术来进行数据的传送，好让网站带宽的可利用率上升喔
- 压缩文件的扩展名大多是：“*.gz, .bz2, .xz, .tar, .tar.gz,* .tar.bz2, *.tar.xz”
- 常见的压缩指令有 gzip, bzip2, xz。压缩率最佳的是 xz，若可以不计时间成本，建议使用 xz 进行压缩。
- tar 可以用来进行文件打包，并可支持 gzip, bzip2, xz 的压缩。
- 压　缩：tar -Jcv -f filename.tar.xz 要被压缩的文件或目录名称
- 查　询：tar -Jtv -f filename.tar.xz
- 解压缩：tar -Jxv -f filename.tar.xz -C 欲解压缩的目录
- xfsdump 指令可备份文件系统或单一目录
- xfsdump 的备份若针对文件系统时，可进行 0-9 的 level 差异备份！其中 level 0 为完整备份；
- xfsrestore 指令可还原被 xfsdump 创建的备份文件；
- 要创建光盘烧录数据时，可通过 mkisofs 指令来创建；
- 可通过 wodim 来写入 CD 或 DVD 烧录机
- dd 可备份完整的 partition 或 disk ，因为 dd 可读取磁盘的 sector 表面数据
- cpio 为相当优秀的备份指令，不过必须要搭配类似 find 指令来读入欲备份的文件名数据，方可进行备份动作。

8.8 本章习题

（要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看）

- 情境仿真题一：请将本章练习过程中产生的不必要的文件删除，以保持系统容量不要被恶搞！
    - rm /home/CentOS-7-x86_64-Minimal-1503-01.iso
    - rm -rf /srv/newcd/
    - rm /custom.iso
    - rm -rf /tmp/vda2.img /tmp/boot.cpio /tmp/boot /tmp/boot2 /tmp/boot3
    - rm -rf /tmp/services *tmp/system.*
    - rm -rf /root/etc* /root/system.tar.bz2 /root/boot
- 情境仿真题二：你想要逐时备份 /home 这个目录内的数据，又担心每次备份的信息太多， 因此想要使用 xfsdump 的方式来逐一备份数据到 /backups 这个目录下。该如何处理？
    - 目标：了解到 xfsdump 以及各个不同 level 的作用；
    - 前提：被备份的数据为单一 partition ，亦即本例中的 /home 实际处理的方法其实还挺简单的！我们可以这样做看看：

    - 先替该目录制作一些数据，亦即复制一些东西过去吧！ mkdir /home/chapter8; cp -a /etc /boot /home/chapter8

    - 开始进行 xfsdump ，记得，一开始是使用 level 0 的完整备份喔！ mkdir /backups xfsdump -l 0 -L home_all -M home_all -f /backups/home.dump /home

    - 尝试将 /home 这个文件系统加大，将 /var/log/ 的数据复制进去吧！ cp -a /var/log/ /home/chapter8 此时原本的 /home 已经被改变了！继续进行备份看看！

    - 将 /home 以 level 1 来进行备份： xfsdump -l 1 -L home_1 -M home_1 -f /backups/home.dump.1 /home ls -l /backups 你应该就会看到两个文件，其中第二个文件 （home.dump.1） 会小的多！这样就搞定啰备份数据！

- 情境仿真三：假设过了一段时间后，你的 /home 变的怪怪的，你想要将该 filesystem 以刚刚的备份数据还原， 此时该如何处理呢？你可以这样做的：

    1. 由于 /home 这个 partition 是用户只要有登陆就会使用，因此你应该无法卸载这个东西！因此，你必须要登出所有一般用户， 然后在 tty2 直接以 root 登陆系统，不要使用一般帐号来登陆后 su 转成 root ！这样才有办法卸载 /home 喔！

    2. 先将 /home 卸载，并且将该 partition 重新格式化！ df -h /home /dev/mapper/centos-home 5.0G 245M 4.8G 5% /home umount /home mkfs.xfs -f /dev/mapper/centos-home

3. 重新挂载原本的 partition ，此时该目录内容应该是空的！ mount -a 你可以自行使用 df 以及 ls -l /home 查阅一下该目录的内容，是空的啦！
4. 将完整备份的 level 0 的文件 /backups/home.dump 还原回来： cd /home xfsrestore -f /backups/home.dump . 此时该目录的内容为第一次备份的状态！还需要进行后续的处理才行！
5. 将后续的 level 1 的备份也还原回来： xfsrestore -f /backups/home.dump.1 . 此时才是恢复到最后一次备份的阶段！如果还有 level 2, level 3 时，就得要一个一个的依序还原才行！
6. 最后删除本章练习的复制档 rm -rf /home/chapter8

8.9 参考资料与延伸阅读

- 台湾学术网络管理文件：Backup Tools in UNIX（Linux）：http://nmc.nchu.edu.tw/tanet/backup_tools_in_unix.htm
- 中文 How to 文件计划（CLDP）：http://www.linux.org.tw/CLDP/HOWTO/hardware/CD-Writing-HOWTO/CD-Writing-HOWTO-3.html
- 熊宝贝工作记录之： Linux 烧录实作：http://csc.ocean-pioneer.com/docum/linux_burn.html
- PHP5 网管实验室： http://www.php5.idv.tw/html.php?mod=article&do=show&shid=26
- CentOS 7.x 之 man xfsdump
- CentOS 7.x 之 man xfsrestore

2003/02/09：第一次完成 2003/05/05：修改 tar 的部分内容，尤其是 -P 这个参数的配合用法 2005/07/26：将旧有的文章移动到 这里 2005/07/27：大略修改了一些风格，另外，动作较大的是在范例的部分！ 2005/08/29：加入了 dd 这个有趣的指令喔！ 2006/05/02：将原本“tar -zxvpf /tmp/etc.tar.gz /etc”修改为“tar -zcvpf /tmp/etc.tar.gz /etc” 感谢讨论区网友 chinu 提供的信息。 2008/10/31：将原本针对 FC4 的旧版本移动到此处 2008/12/18：这次的改版在这一章添加了不少东西！尤其是将 cpio 与 dd 的范例重新做个整理！并加入 dump/restore, mkisofs/cdrecord 2009/08/20：加入情境仿真的题目。 2015/06/29：将就的基于 CentOS 5.x 的版本移动到此处 2015/06/29：因为 compress 几乎没人在用了，所以这一章将他拿掉，而加入的是新的 xz 喔！ 2015/07/16：竟然忘记复原 /etc 需要恢复 SELinux type 的说明！请参考这一个项目的说明喔！

第九章、vim 程序编辑器

最近更新日期：20//

系统管理员的重要工作就是得要修改与设置某些重要软件的配置文件，因此至少得要学会一种以上的命令行的文书编辑器。 在所有的 Linux distributions 上头都会有的一套文书编辑器就是 vi ，而且很多软件默认也是使用 vi 做为他们编辑的接口， 因此鸟哥建议您务必要学会使用 vi 这个正规的文书编辑器。此外，vim 是进阶版的 vi ， vim 不但可以用不同颜色显示文字内容，还能够进行诸如 shell script, C program 等程序编辑功能， 你可以将 vim 视为一种程序编辑器！鸟哥也是用 vim 编辑鸟站的网页文章呢！ ^_^

9.1 vi 与 vim

由前面一路走来，我们一直建议使用文字模式来处理 Linux 系统的设置问题，因为不但可以让你比较容易了解到 Linux 的运行状况，也比较容易了解整个设置的基本精神，更能“保证”你的修改可以顺利的被运行。 所以，在 Linux 的系统中使用文字编辑器来编辑你的 Linux 参数配置文件，可是一件很重要的事情呦！也因此呢，系统管理员至少应该要熟悉一种文书处理器的！



Tips 这里要再次的强调，不同的 Linux distribution 各有其不同的附加软件，例如 Red Hat Enterprise Linux 与 Fedora 的 ntsysv 与 setup 等，而 SuSE 则有 YAST 管理工具等等， 因此，如果你只会使用此种类型的软件来控制你的 Linux 系统时，当接管不同的 Linux distributions 时，呵呵！那可就苦恼了！

在 Linux 的世界中，绝大部分的配置文件都是以 ASCII 的纯文本形态存在，因此利用简单的文字编辑软件就能够修改设置了！ 与微软的 Windows 系统不同的是，如果你用惯了 Microsoft Word 或 Corel Wordperfect 的话，那么除了 X window 里面的图形接口编辑程序（如 xemacs ）用起来尚可应付外，在 Linux 的文字模式下，会觉得文书编辑程序都没有窗口接口来的直观与方便。



Tips 什么是纯文本文件？其实文件记录的就是 0 与 1 ，而我们通过编码系统来将这些 0 与 1 转成我们认识的文字就是了。 在第零章里面的数据表示方式有较多说明，请自行查阅。 ASCII 就是其中一种广为使用的文字编码系统，在 ASCII 系统中的图示与代码可以参考 http://zh.wikipedia.org/wiki/ASCII呢！

那么 Linux 在命令行下的文书编辑器有哪些呢？其实有非常多喔！常常听到的就有： emacs, pico, nano, joe, 与 vim 等等[1]。 既然有这么多命令行的文书编辑器，那么我们为什么一定要学 vi 啊？还有那个 vim 是做啥用的？下面就来谈一谈先！

9.1.1 为何要学 vim

文书编辑器那么多，我们之前在第四章也曾经介绍过那简单好用的 nano ，既然已经学会了 nano ，干嘛鸟哥还一直要你学这不是很友善的 vi 呢？其实是有原因的啦！因为：

- 所有的 Unix Like 系统都会内置 vi 文书编辑器，其他的文书编辑器则不一定会存在；
- 很多个别软件的编辑接口都会主动调用 vi （例如未来会谈到的 crontab, visudo, edquota 等指令）；
- vim 具有程序编辑的能力，可以主动的以字体颜色辨别语法的正确性，方便程序设计；
- 因为程序简单，编辑速度相当快速。

其实重点是上述的第二点，因为有太多 Linux 上面的指令都默认使用 vi 作为数据编辑的接口，所以你必须、一定要学会 vi ，否则很多指令你根本就无法操作呢！这样说，有刺激到你务必要学会 vi 的热情了吗？ ^_^

那么什么是 vim 呢？其实你可以将 vim 视作 vi 的进阶版本，vim 可以用颜色或底线等方式来显示一些特殊的信息。 举例来说，当你使用 vim 去编辑一个 C 程序语言的文件，或者是我们后续会谈到的 shell script 脚本程序时，vim 会依据文件的扩展名或者是文件内的开头信息，判断该文件的内容而自动的调用该程序的语法判断式，再以颜色来显示程序码与一般信息。也就是说， 这个 vim 是个“程序编辑器”啦！甚至一些 Linux 基础配置文件内的语法，都能够用 vim 来检查呢！ 例如我们在第七章谈到的 /etc/fstab 这个文件的内容。

简单的来说， vi 是老式的文书处理器，不过功能已经很齐全了，但是还是有可以进步的地方。 vim 则可以说是程序开发者的一项很好用的工具，就连 vim 的官方网站 （http://www.vim.org） 自己也说 vim 是一个“程序开发工具”而不是文书处理软件～^_^。 因为 vim 里面加入了很多额外的功能，例如支持正则表达式的搜寻架构、多文件编辑、区块复制等等。 这对于我们在 Linux 上面进行一些配置文件的修订工作时，是很棒的一项功能呢！



Tips 什么时候会使用到 vim 呢？其实鸟哥的整个网站都是在 vim 的环境下一字一字的创建起来的喔！ 早期鸟哥使用网页制作软件在编写网页，但是老是发现网页编辑软件都不怎么友善，尤其是写到 PHP 方面的程序码时。 后来就干脆不使用所见即所得的编辑软件，直接使用 vim ，然后标签 （tag） 也都自行用键盘输入！ 这样整个文件也比较干净！所以说，鸟哥我是很喜欢 vim 的啦！ ^_^

下面鸟哥会先就简单的 vi 做个介绍，然后再跟大家报告一下 vim 的额外功能与用法呢！

9.2 vi 的使用

基本上 vi 共分为三种模式，分别是“一般指令模式”、“编辑模式”与“命令行命令模式”。 这三种模式的作用分别是：

- 一般指令模式 （command mode）

以 vi 打开一个文件就直接进入一般指令模式了（这是默认的模式，也简称为一般模式）。在这个模式中， 你可以使用“上下左右”按键来移动光标，你可以使用“删除字符”或“删除整列”来处理文件内容， 也可以使用“复制、贴上”来处理你的文件数据。

- 编辑模式 （insert mode）

在一般指令模式中可以进行删除、复制、贴上等等的动作，但是却无法编辑文件内容的！ 要等到你按下“i, I, o, O, a, A, r, R”等任何一个字母之后才会进入编辑模式。注意了！通常在 Linux 中，按下这些按键时，在画面的左下方会出现“ INSERT 或 REPLACE ”的字样，此时才可以进行编辑。而如果要回到一般指令模式时， 则必须要按下“Esc”这个按键即可退出编辑模式。

- 命令行命令模式 （command-line mode）

在一般模式当中，输入“ : / ? ”三个中的任何一个按钮，就可以将光标移动到最下面那一列。在这个模式当中， 可以提供你“搜寻数据”的动作，而读取、存盘、大量取代字符、离开 vi 、显示行号等等的动作则是在此模式中达成的！

简单的说，我们可以将这三个模式想成下面的图示来表示：

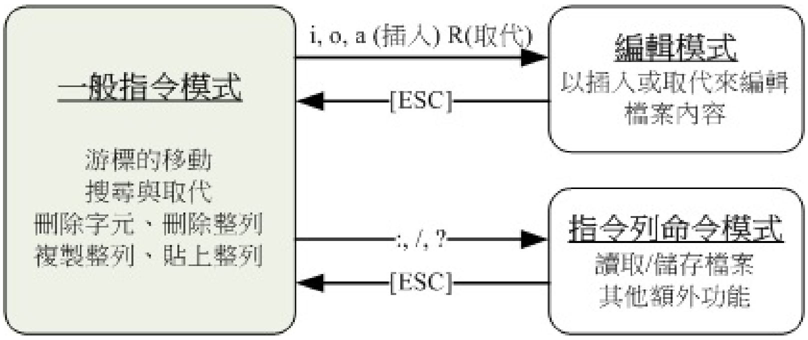


图9.2.1、vi三种模式的相互关系

注意到上面的图示，你会发现一般指令模式可与编辑模式及命令行界面切换， 但编辑模式与命令行界面之间不可互相切换喔！这非常重要啦！闲话不多说，我们下面以一个简单的例子来进行说明吧！



Tips 过去鸟哥的前一版本中，一般指令模式被称为一般模式。但是英文版的 vi/vim 说明中，一般模式其实是“ command mode ”的意思！中文直译会变成指令模式啊！ 之所以称为指令模式，主因是我们可以在一般模式下面按下很多特殊的指令功能！例如删除、复制、区块选择等等！只是这个模式很容易跟命令行界面 （command-line） 混淆～ 所以鸟哥过去才称为一般模式而已。不过真的很容易误解啦！所以这一版开始，这一模式被鸟哥改为“一般指令模式”了！要尊重英文原文！

9.2.1 简易执行范例

如果你想要使用 vi 来创建一个名为 welcome.txt 的文件时，你可以这样做：

- 1. 使用“ vi filename ”进入一般指令模式

```
[dmtsai@study ~]$ /bin/vi welcome.txt
# 在 CentOS 7 当中，由于一般帐号默认 vi 已经被 vim 取代了，因此得要输入绝对路径来执行才行！
```

直接输入“ vi 文件名”就能够进入 vi 的一般指令模式了。不过请注意，由于一般帐号默认已经使用 vim 来取代，因此如上表所示， 如果使用一般帐号来测试，得要使用绝对路径的方式来执行 /bin/vi 才好！另外，请注意，记得 vi 后面一定要加文件名，不管该文件名存在与否！

整个画面主要分为两部份，上半部与最下面一列两者可以视为独立的。如下图 9.2.2 所示，图中那个虚线是不存在的， 鸟哥用来说明而已啦！上半部显示的是文件的实际内容，最下面一列则是状态显示列（如下图的[New File]信息），或者是命令下达列喔！

图9.2.2、用 vi 打开一个新文件

如果你打开的文件是旧文件（已经存在的文件），则可能会出现如下的信息：

图9.2.3、用 vi 打开一个旧文件

如上图 9.2.3 所示，箭头所指的那个“"/etc/man_db.conf" [readonly] 131L, 5171C”代表的是“现在打开的文件名为 /etc/man_db.conf， 由于启动者的身份缘故，目前文件为只读状态，且文件内有 131 列 以及具有 5171 个字符”的意思！那一列的内容并不是在文件内，而是 vi 显示一些信息的地方喔！此时是在一般指令模式的环境下啦。接下来开始来输入吧！

- 2. 按下 i 进入编辑模式，开始编辑文字

在一般指令模式之中，只要按下 i, o, a 等字符就可以进入编辑模式了！在编辑模式当中，你可以发现在左下角状态列中会出现 –INSERT- 的字样，那就是可以输入任意字符的提示啰！这个时候，键盘上除了 [Esc] 这个按键之外，其他的按键都可以视作为一般的输入按钮了，所以你可以进行任何的编辑啰！

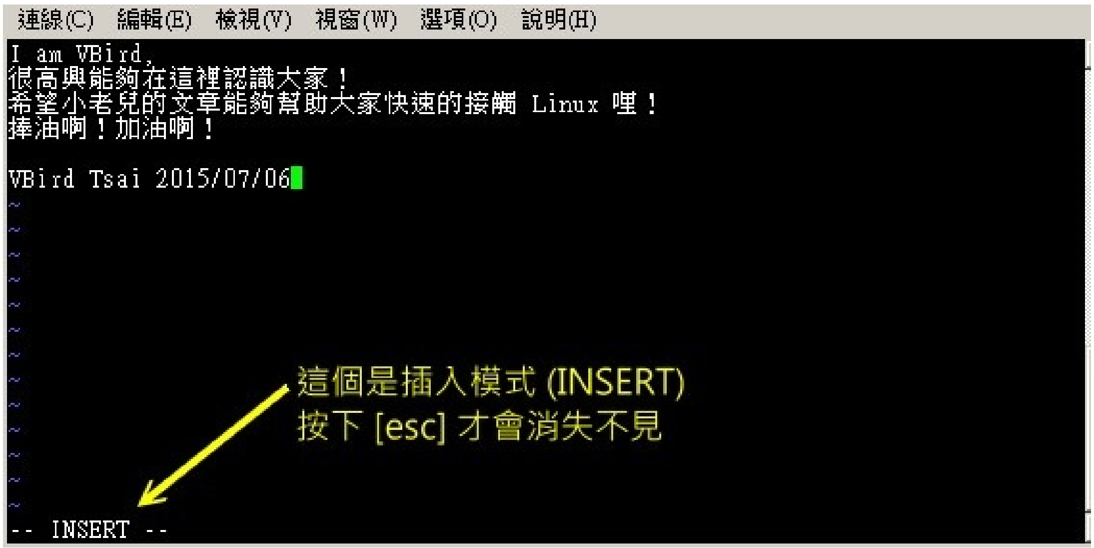


图9.2.4、开始用 vi 来进行编辑

- 3. 按下 [ESC] 按钮回到一般指令模式

好了，假设我已经按照上面的样式给他编辑完毕了，那么应该要如何退出呢？是的！没错！就是给他按下 [Esc] 这个按钮即可！马上你就会发现画面左下角的 – INSERT – 不见了！！

- 4. 进入命令行界面，文件储存并离开 vi 环境

OK，我们要存盘了，存盘 （write） 并离开 （quit） 的指令很简单，输入“:wq”即可存盘离开！ （注意了，按下 : 该光标就会移动到最下面一列去！） 这时你在提示字符后面输入“ ls -l ”即可看到我们刚刚创建的 welcome.txt 文件啦！整个图示有点像下面这样：

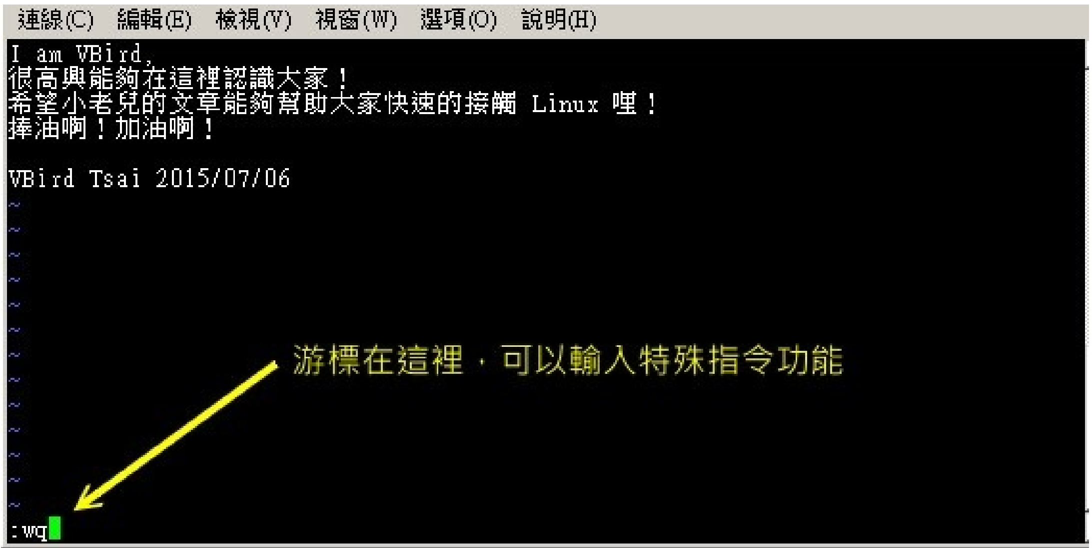


图9.2.5、在命令行界面进行储存及离开 vi 环境

如此一来，你的文件 welcome.txt 就已经创建起来啰！需要注意的是，如果你的文件权限不对，例如为 -r--r--r-- 时，那么可能会无法写入，此时可以使用“强制写入”的方式吗？可以！使用“ :wq! ” 多加一个惊叹号即可！不过，需要特别注意呦！那个是在“你的权限可以改变”的情况下才能成立的！ 关于权限的概念，请自行回去翻一下第五章的内容吧！

9.2.2 按键说明

除了上面简易范例的 i, [Esc], :wq 之外，其实 vi 还有非常多的按键可以使用喔！在介绍之前还是要再次强调， vi 的三种模式只有一般指令模式可以与编辑、命令行界面切换，编辑模式与命令行界面之间并不能切换的！ 这点在图9.2.1里面有介绍到，注意去看看喔！下面就来谈谈 vi 软件中会用到的按键功能吧！

- 第一部份：一般指令模式可用的按钮说明，光标移动、复制贴上、搜寻取代等

移动光标的方法	
h 或 向左方向键（←）	光标向左移动一个字符
j 或 向下方向键（↓）	光标向下移动一个字符
k 或 向上方向键（↑）	光标向上移动一个字符
l 或 向右方向键（→）	光标向右移动一个字符
如果你将右手放在键盘上的话，你会发现 hjkl 是排列在一起的，因此可以使用这四个按钮来移动光标。 如果想要进行多次移动的话，例如向下移动 30 列，可以使用 "30j" 或 "30↓" 的组合按键，亦即加上想要进行的次数（数字）后，按下动作即可！	
[Ctrl] + [f]	屏幕“向下”移动一页，相当于 [Page Down]按键 （常用）
[Ctrl] + [b]	屏幕“向上”移动一页，相当于 [Page Up] 按键（常用）
[Ctrl] + [d]	屏幕“向下”移动半页
[Ctrl] + [u]	屏幕“向上”移动半页
+	光标移动到非空白字符的下一列
-	光标移动到非空白字符的上一列
n<space>	那个 n 表示“数字”，例如 20 。按下数字后再按空白键，光标会向右移动这一列的 n 个字符。例如 20<space> 则光标会向后面移动 20 个字符距离。
0 或功能键[Home]	这是数字“ 0 ”：移动到这一列的最前面字符处（常用）
$ 或功能键[End]	移动到这一列的最后面字符处（常用）
H	光标移动到这个屏幕的最上方那一列的第一个字符
M	光标移动到这个屏幕的中央那一列的第一个字符

L	光标移动到这个屏幕的最下方那一列的第一个字符
G	移动到这个文件的最后一列（常用）
nG	n 为数字。移动到这个文件的第 n 列。例如 20G 则会移动到这个文件的第 20 列（可配合 :set nu）
gg	移动到这个文件的第一列，相当于 1G 啊！（常用）
n<Enter>	n 为数字。光标向下移动 n 列（常用）
搜寻与取代	
/word	向光标之下寻找一个名称为 word 的字串。例如要在文件内搜寻 vbird 这个字串，就输入 /vbird 即可！（常用）
?word	向光标之上寻找一个字串名称为 word 的字串。
n	这个 n 是英文按键。代表“<u>重复前一个搜寻的动作</u>”。举例来说， 如果刚刚我们执行 /vbird 去向下搜寻 vbird 这个字串，则按下 n 后，会向下继续搜寻下一个名称为 vbird 的字串。如果是执行 ?vbird 的话，那么按下 n 则会向上继续搜寻名称为 vbird 的字串！
N	这个 N 是英文按键。与 n 刚好相反，为“反向”进行前一个搜寻动作。 例如 /vbird 后，按下 N 则表示“向上”搜寻 vbird 。
使用 /word 配合 n 及 N 是非常有帮助的！可以让你重复的找到一些你搜寻的关键字！	
:n1,n2s/word1/word2/g	n1 与 n2 为数字。在第 n1 与 n2 列之间寻找 word1 这个字串，并将该字串取代为 word2 ！举例来说，在 100 到 200 列之间搜寻 vbird 并取代为 VBIRD 则：“:100,200s/vbird/VBIRD/g”。（常用）
:1,$s/word1/word2/g	从第一列到最后一列寻找 word1 字串，并将该字串取代为 word2 ！（常用）
:1,$s/word1/word2/gc	从第一列到最后一列寻找 word1 字串，并将该字串取代为 word2 ！且在取代前显示提示字符给使用者确认 （confirm） 是否需要取代！（常用）
删除、复制与贴上	
x, X	在一列字当中，x 为向后删除一个字符 （相当于 [del] 按键）， X 为向前删除一个字符（相当于 [backspace] 亦即是倒退键） （常用）

nx	n 为数字，连续向后删除 n 个字符。举例来说，我要连续删除 10 个字符， “10x”。
dd	删除光标所在的那一整列（常用）
ndd	n 为数字。删除光标所在的向下 n 列，例如 20dd 则是删除 20 列 （常用）
d1G	删除光标所在到第一列的所有数据
dG	删除光标所在到最后一列的所有数据
d$	删除光标所在处，到该列的最后一个字符
d0	那个是数字的 0 ，删除光标所在处，到该列的最前面一个字符
yy	复制光标所在的那一列（常用）
nyy	n 为数字。复制光标所在的向下 n 列，例如 20yy 则是复制 20 列（常用）
y1G	复制光标所在列到第一列的所有数据
yG	复制光标所在列到最后一列的所有数据
y0	复制光标所在的那个字符到该列行首的所有数据
y$	复制光标所在的那个字符到该列行尾的所有数据
p, P	p 为将已复制的数据在光标下一列贴上，P 则为贴在光标上一列！ 举例来说，我目前光标在第 20 列，且已经复制了 10 列数据。则按下 p 后， 那 10 列数据会贴在原本的 20 列之后，亦即由 21 列开始贴。但如果是按下 P 呢？ 那么原本的第 20 列会被推到变成 30 列。 （常用）
J	将光标所在列与下一列的数据结合成同一列
c	重复删除多个数据，例如向下删除 10 列，[10cj]
u	复原前一个动作。（常用）
[Ctrl]+r	重做上一个动作。（常用）
这个 u 与 [Ctrl]+r 是很常用的指令！一个是复原，另一个则是重做一次～ 利用这两个功能按键，你的编辑，嘿嘿！很快乐的啦！	
.	不要怀疑！这就是小数点！意思是重复前一个动作的意思。 如果你想要重复删除、重复贴上等等动作，按下小数点“.”就好了！ （常用）

- 第二部份：一般指令模式切换到编辑模式的可用的按钮说明

进入插入或取代的编辑模式	
i, I	进入插入模式（Insert mode）：i 为“从目前光标所在处插入”， I 为“在目前所在列的第一个非空白字符处开始插入”。（常用）
a, A	进入插入模式（Insert mode）：a 为“从目前光标所在的下一个字符处开始插入”， A 为“从光标所在列的最后一个字符处开始插入”。（常用）
o, O	进入插入模式（Insert mode）：这是英文字母 o 的大小写。o 为“在目前光标所在的下一列处插入新的一列”； O 为在目前光标所在处的上一列插入新的一列！（常用）
r, R	进入取代模式（Replace mode）：r 只会取代光标所在的那一个字符一次；R会一直取代光标所在的文字，直到按下 ESC 为止；（常用）
上面这些按键中，在 vi 画面的左下角处会出现“--INSERT--”或“--REPLACE--”的字样。 由名称就知道该动作了吧！！特别注意的是，我们上面也提过了，你想要在文件里面输入字符时， 一定要在左下角处看到 INSERT 或 REPLACE 才能输入喔！	
[Esc]	退出编辑模式，回到一般指令模式中（常用）

- 第三部份：一般指令模式切换到命令行界面的可用按钮说明

命令行界面的储存、离开等指令	
:w	将编辑的数据写入硬盘文件中（常用）
:w!	若文件属性为“只读”时，强制写入该文件。不过，到底能不能写入， 还是跟你对该文件的文件权限有关啊！
:q	离开 vi （常用）
:q!	若曾修改过文件，又不想储存，使用！为强制离开不储存盘案。
注意一下啊，那个惊叹号（!） 在 vi 当中，常常具有“强制”的意思～	
:wq	储存后离开，若为 :wq! 则为强制储存后离开 （常用）
ZZ	这是大写的 Z 喔！若文件没有更动，则不储存离开，若文件已经被更动过，则储存后离开！
:w [filename]	将编辑的数据储存成另一个文件（类似另存新文件）
:r [filename]	在编辑的数据中，读入另一个文件的数据。亦即将 “filename” 这个文件内容加到光标所在列后面
:n1,n2 w [filename]	将 n1 到 n2 的内容储存成 filename 这个文件。
:! command	暂时离开 vi 到命令行界面下执行 command 的显示结果！例如 “:! ls /home”即可在 vi 当中察看 /home 下面以 ls 输出的文件信息！
vim 环境的变更	
:set nu	显示行号，设置之后，会在每一列的字首显示该列的行号
:set nonu	与 set nu 相反，为取消行号！

特别注意，在 vi 中，“数字”是很有意义的！数字通常代表重复做几次的意思！ 也有可能是代表去到第几个什么什么的意思。举例来说，要删除 50 列，则是用 “50dd” 对吧！ 数字加在动作之前～那我要向下移动 20 列呢？那就是“20j”或者是“20↓”即可。

OK！会这些指令就已经很厉害了，因为常用到的指令也只有不到一半！通常 vi 的指令除了上面鸟哥注明的常用的几个外，其他是不用背的，你可以做一张简单的指令表在你的屏幕墙上， 一有疑问可以马上的查询呦！这也是当初鸟哥使用 vim 的方法啦！

9.2.3 一个案例练习

来来来！赶紧测试一下你是否已经熟悉 vi 这个指令呢？请依照下面的需求进行指令动作。（下面的操作为使用 CentOS 7.1 中的 man_db.conf 来做练习的，该文件你可以在这里下载： http://linux.vbird.org/linux_basic/0310vi/man_db.conf。） 看看你的显示结果与鸟哥的结

果是否相同啊？

1. 请在 /tmp 这个目录下创建一个名为 vitest 的目录；
2. 进入 vitest 这个目录当中；
3. 将 /etc/man_db.conf 复制到本目录下面（或由上述的链接下载 man_db.conf 文件）；
4. 使用 vi 打开本目录下的 man_db.conf 这个文件；
5. 在 vi 中设置一下行号；
6. 移动到第 43 列，向右移动 59 个字符，请问你看到的小括号内是哪个文字？
7. 移动到第一列，并且向下搜寻一下“ gzip ”这个字串，请问他在第几列？
8. 接着下来，我要将 29 到 41 列之间的“小写 man 字串”改为“大写 MAN 字串”，并且一个一个挑选是否需要修改，如何下达指令？如果在挑选过程中一直按“y”， 结果会在最后一列出现改变了几个 man 呢？
9. 修改完之后，突然反悔了，要全部复原，有哪些方法？
10. 我要复制 66 到 71 这 6 列的内容（含有MANDB_MAP），并且贴到最后一列之后；
11. 113 到 128 列之间的开头为 # 符号的注解数据我不要了，要如何删除？
12. 将这个文件另存成一个 man.test.config 的文件名；
13. 去到第 25 列，并且删除 15 个字符，结果出现的第一个单字是什么？
14. 在第一列新增一列，该列内容输入“I am a student...”；
15. 储存后离开吧！

整个步骤可以如下显示：

1. “mkdir /tmp/vitest”
2. “cd /tmp/vitest”
3. “cp /etc/man_db.conf .”
4. “/bin/vi man_db.conf”
5. “:set nu”然后你会在画面中看到左侧出现数字即为行号。
6. 先按下“43G”再按下“59→”会看到“ as ”这个单字在小括号内；
7. 先执行“1G”或“gg”后，直接输入“/gzip”，则会去到第 93 列才对！
8. 直接下达“ :29,41s/man/MAN/gc ”即可！若一直按“y”最终会出现“在 13 列内置换 13 个字串”的说明。
9. （1）简单的方法可以一直按“ u ”回复到原始状态，（2）使用不储存离开“ :q! ”之后，再重新读取一次该文件；
10. “66G” 然后再“ 6yy ”之后最后一列会出现“复制6列”之类的说明字样。 按下“ G ”到最后一列，再给他“ p ”贴上6列！
11. 因为 113~128 共 16 列，因此“ 113G ”→“ 16dd ”就能删除 16 列，此时你会发现光标所在 113 列的地方变成 “ # Flags. ” 开头啰
12. “ :w man.test.config ”，你会发现最后一列出现 "man.test.config" [New].. 的字样。
13. “25G” 之后，再给他“ 15x ”即可删除 15 个字符，出现“ tree ”的字样；
14. 先“ 1G ”去到第一列，然后按下大写的“ O ”便新增一列且在插入模式；开始输入“I am a student...”后， 按下[Esc]回到一般指令模式等待后续工作；
15. “:wq”

如果你的结果都可以查的到，那么 vi 的使用上面应该没有太大的问题啦！剩下的问题会是在…打字练习…。

9.2.4 vim 的暂存盘、救援回复与打开时的警告讯息

在目前主要的文书编辑软件都会有“回复”的功能，亦即当你的系统因为某些原因而导致类似死机的情况时， 还可以通过某些特别的机制来让你将之前未储存的数据“救”回来！这就是鸟哥这里所谓的“回复”功能啦！ 那么 vim 有没有回复功能呢？有的！ vim 就是通过“暂存盘”来救援的啦！

当我们在使用 vim 编辑时， vim 会在与被编辑的文件的目录下，再创建一个名为 .filename.swp 的文件。 比如说我们在上一个小节谈到的编辑 /tmp/vitest/man_db.conf 这个文件时， vim 会主动的创建 /tmp/vitest/.man_db.conf.swp 的暂存盘，你对 man_db.conf 做的动作就会被记录到这个 .man_db.conf.swp 当中喔！如果你的系统因为某些原因断线了， 导致你编辑的文件还没有储存，这个时候 .man_db.conf.swp 就能够发挥救援的功能了！我们来测试一下吧！ 下面的练习有些部分的指令我们尚未谈到，没关系，你先照着做，后续再回来了解啰！

```
[dmtsai@study ~]$ cd /tmp/vitest
[dmtsai@study vitest]$ vim man_db.conf
# 此时会进入到 vim 的画面，请在 vim 的一般指令模式下按下“ [ctrl]-z ”的组合键

[1]+  Stopped             vim man_db.conf  &lt;==按下 [ctrl]-z 会告诉你这个讯息
```

当我们在 vim 的一般指令模式下按下 [ctrl]-z 的组合按键时，你的 vim 会被丢到背景去执行！ 这部份的功能我们会在第十六章的程序管理当中谈到， 你这里先知道一下即可。回到命令提示字符后，接下来我们来仿真将 vim 的工作不正常的中断吧！

```
[dmtsai@study vitest]$ ls -al
drwxrwxr-x.  2 dmtsai dmtsai    69 Jul  6 23:54 .
drwxrwxrwt. 17 root   root    4096 Jul  6 23:53 ..
-rw-r--r--.  1 dmtsai dmtsai  4850 Jul  6 23:47 man_db.conf
-rw-r--r--.  1 dmtsai dmtsai 16384 Jul  6 23:54 .man_db.conf.swp  &lt;==就是他，暂存盘
-rw-rw-r--.  1 dmtsai dmtsai  5442 Jul  6 23:35 man.test.config

[dmtsai@study vitest]$ kill -9 %1 &lt;==这里仿真断线停止 vim 工作
[dmtsai@study vitest]$ ls -al .man_db.conf.swp
-rw-r--r--. 1 dmtsai dmtsai 16384 Jul  6 23:54 .man_db.conf.swp  &lt;==暂存盘还是会存在！
```

那个 kill 可以仿真将系统的 vim 工作删除的情况，你可以假装死机了啦！ 由于 vim 的工作被不正常的中断，导致暂存盘无法借由正常流程来结束， 所以暂存盘就不会消失，而继续保留下来。此时如果你继续编辑那个 man_db.conf ，会出现什么情况呢？ 会出现如下所示的状态喔：

```
[dmtsai@study vitest]$ vim man_db.conf

E325: ATTENTION  &lt;==错误代码
Found a swap file by the name ".man_db.conf.swp"  &lt;==下面数列说明有暂存盘的存在
          owned by: dmtsai   dated: Mon Jul  6 23:54:16 2015
         file name: /tmp/vitest/man_db.conf  &lt;==这个暂存盘属于哪个实际的文件？
          modified: no
         user name: dmtsai   host name: study.centos.vbird
        process ID: 31851
While opening file "man_db.conf"
             dated: Mon Jul  6 23:47:21 2015

下面说明可能发生这个错误的两个主要原因与解决方案！
（1） Another program may be editing the same file.  If this is the case,
    be careful not to end up with two different instances of the same
    file when making changes.  Quit, or continue with caution.
（2） An edit session for this file crashed.
    If this is the case, use ":recover" or "vim -r man_db.conf"
    to recover the changes （see ":help recovery"）.
    If you did this already, delete the swap file ".man_db.conf.swp"
    to avoid this message.

Swap file ".man_db.conf.swp" already exists! 下面说明你可进行的动作
[O]pen Read-Only, （E）dit anyway, （R）ecover, （D）elete it, （Q）uit, （A）bort:
```

由于暂存盘存在的关系，因此 vim 会主动的判断你的这个文件可能有些问题，在上面的图示中 vim 提示两点主要的问题与解决方案，分别是这样的：

- 问题一：可能有其他人或程序同时在编辑这个文件：

  由于 Linux 是多用户多任务的环境，因此很可能有很多人同时在编辑同一个文件。如果在多人共同编辑的情况下， 万一大家同时储存，那么这个文件的内容将会变的乱七八糟！为了避免这个问题，因此 vim 会出现这个警告窗口！ 解决的方法则是：

  - 找到另外那个程序或人员，请他将该 vim 的工作结束，然后你再继续处理。
  - 如果你只是要看该文件的内容并不会有任何修改编辑的行为，那么可以选择打开成为只读（O）文件， 亦即上述画面反白部分输入英文“ o ”即可，其实就是 [O]pen Read-Only 的选项啦！

- 问题二：在前一个 vim 的环境中，可能因为某些不知名原因导致 vim 中断（crashed）：

  这就是常见的不正常结束 vim 产生的后果。解决方案依据不同的情况而不同喔！常见的处理方法为：

  - 如果你之前的 vim 处理动作尚未储存，此时你应该要按下“R”，亦即使用 （R）ecover 的项目， 此时 vim 会载入 .man_db.conf.swp 的内容，让你自己来决定要不要储存！这样就能够救回来你之前未储存的工作。 不过那个 .man_db.conf.swp 并不会在你结束 vim 后自动删除，所以你离开 vim 后还得要自行删除 .man_db.conf.swp 才能避免每次打开这个文件都会出现这样的警告！

  - 如果你确定这个暂存盘是没有用的，那么你可以直接按下“D”删除掉这个暂存盘，亦即 （D）elete it 这个项目即可。 此时 vim 会载入 man_db.conf ，并且将旧的 .man_db.conf.swp 删除后，创建这次会使用的新的 .man_db.conf.swp 喔！

至于这个发现暂存盘警告讯息的画面中，有出现六个可用按钮，各按钮的说明如下：

- [O]pen Read-Only：打开此文件成为只读文件， 可以用在你只是想要查阅该文件内容并不想要进行编辑行为时。一般来说，在上课时，如果你是登陆到同学的计算机去看他的配置文件， 结果发现其实同学他自己也在编辑时，可以使用这个模式；
- （E）dit anyway：还是用正常的方式打开你要编辑的那个文件， 并不会载入暂存盘的内容。不过很容易出现两个使用者互相改变对方的文件等问题！不好不好！
- （R）ecover：就是载入暂存盘的内容，用在你要救回之前未储存的工作。 不过当你救回来并且储存离开 vim 后，还是要手动自行删除那个暂存盘喔！
- （D）elete it：你确定那个暂存盘是无用的！那么打开文件前会先将这个暂存盘删除！ 这个动作其实是比较常做的！因为你可能不确定这个暂存盘是怎么来的，所以就删除掉他吧！哈哈！
- （Q）uit：按下 q 就离开 vim ，不会进行任何动作回到命令提示字符。
- （A）bort：忽略这个编辑行为，感觉上与 quit 非常类似！ 也会送你回到命令提示字符就是啰！

9.3 vim 的额外功能

其实，目前大部分的 distributions 都以 vim 取代 vi 的功能了！如果你使用 vi 后，却看到画面的右下角有显示目前光标所在的行列号码，那么你的 vi 已经被 vim 所取代啰～为什么要用 vim 呢？因为 vim 具有颜色显示的功能，并且还支持许多的程序语法（syntax），因此，当你使用 vim 编辑程序时（不论是 C 语言，还是 shell script），我们的 vim 将可帮你直接进行“程序除错（debug）”的功能！真的很不赖吧！^_^

如果你在文字模式下，输入 alias 时，出现这样的画面：

```
[dmtsai@study ~]$ alias
....其他省略....
alias vi='vim'  &lt;==重点在这列啊！
```

这表示当你使用 vi 这个指令时，其实就是执行 vim 啦！如果你没有这一列，那么你就必须要使用 vim filename 来启动 vim 啰！基本上，vim 的一般用法与 vi 完全一模一样～没有不同啦！那么我们就来看看 vim 的画面是怎样啰！假设我想要编辑 /etc/services，则输入“vim /etc/services”看看吧：



图 9.3.1、使用 vim 编辑系统配置文件的示范

上面是 vim 的画面示意图，在这个画面中有几点特色要说明喔：

1. 由于 /etc/services 是系统规划的配置文件，因此 vim 会进行语法检验，所以你会看到画面中内部主要为深蓝色，且深蓝色那一列是以注解符号（#）为开头；
2. 画面中的最下面一列，在左边显示该文件的属性，包括只读文件、内容共有 11176 列与 670293 个字符；
3. 最下面一列的右边出现的 1,1 表示光标所在为第一列，第一个字符位置之意（请看上图中的光标所在）；

所以，如果你向下移动到其他位置时，出现的非注解的数据就会有点像这样：



图 9.3.2、使用 vim 编辑系统配置文件的示范

看到了喔！除了注解之外，其他的列就会有特别的颜色显示呢！可以避免你打错字啊！而且， 最右下角的 1% 代表目前这个画面占整体文件的 1% 之意！这样瞭乎？

9.3.1 区块选择（Visual Block）

刚刚我们提到的简单的 vi 操作过程中，几乎提到的都是以列为单位的操作。那么如果我想要搞定的是一个区块范围呢？ 举例来说，像下面这种格式的文件：

```
192.168.1.1    host1.class.net
192.168.1.2    host2.class.net
192.168.1.3    host3.class.net
192.168.1.4    host4.class.net
.....中间省略......
```

这个文件我将他放置到 http://linux.vbird.org/linux_basic/0310vi/hosts ，你可以自行下载来看一看这个文件啊！现在我们来玩一玩这个文件吧！假设我想要将 host1, host2... 等等复制起来， 并且加到每一列的后面，亦即每一列的结果要是“ 192.168.1.2 host2.class.net host2 ”这样的情况时， 在传统或现代的窗口型编辑器似乎不容易达到这个需求，但是咱们的 vim 是办的到的喔！那就使用区块选择 （Visual Block） 吧！当我们按下 v 或者 V 或者 [Ctrl]+v 时，这个时候光标移动过的地方就会开始反白，这三个按键的意义分别是：

区块选择的按键意义	
v	字符选择，会将光标经过的地方反白选择！
V	列选择，会将光标经过的列反白选择！
[Ctrl]+v	区块选择，可以用长方形的方式选择数据
y	将反白的地方复制起来
d	将反白的地方删除掉
p	将刚刚复制的区块，在光标所在处贴上！

来实际进行我们需要的动作吧！就是将 host 再加到每一列的最后面，你可以这样做：

1. 使用 vim hosts 来打开该文件，记得该文件请由上述的链接下载先！
2. 将光标移动到第一列的 host 那个 h 上头，然后按下 [ctrl]-v ，左下角出现区块示意字样：

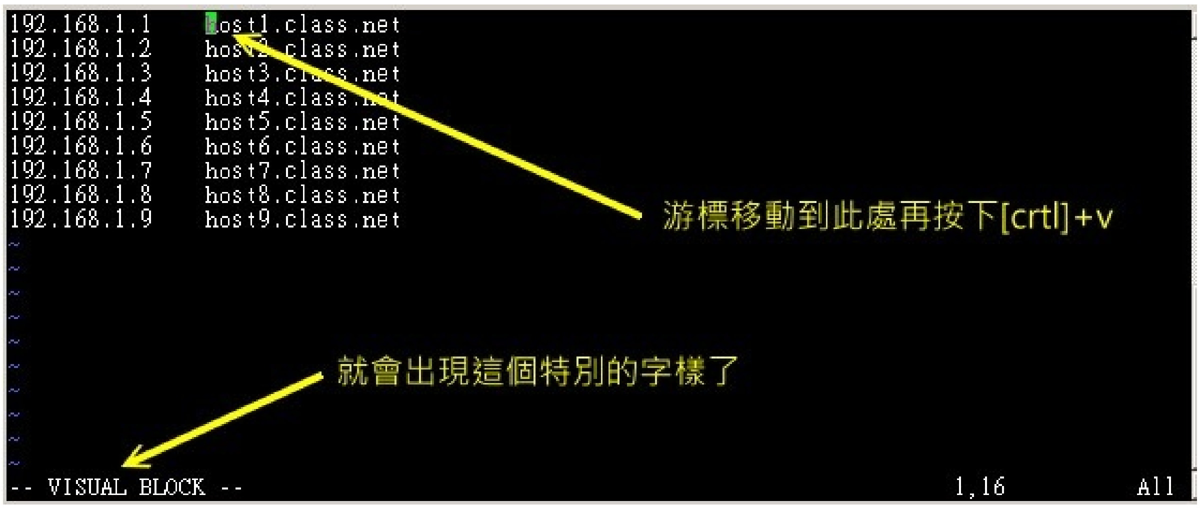


图9.3.3、vim 的区块选择、复制、贴上等功能操作

3. 将光标移动到最底部，此时光标移动过的区域会反白！如下图所示：

图9.3.4、vim 的区块选择、复制、贴上等功能操作

4. 此时你可以按下“ y ”来进行复制，当你按下 y 之后，反白的区块就会消失不见啰！
5. 最后，将光标移动到第一列的最右边，并且再用编辑模式向右按两个空白键，回到一般指令模式后， 再按下“ p ”后，你会发现很有趣！如下图所示：



图9.3.5、vim 的区块选择、复制、贴上等功能操作

通过上述的功能，你可以复制一个区块，并且是贴在某个“区块的范围”内，而不是以列为单位来处理你的整份文件喔！ 鸟哥个人是觉得这玩意儿非常的有帮助啦！至少在进行排列整齐的文字文件中复制/删除区块时，会是一个非常棒的功能！

9.3.2 多文件编辑

假设一个例子，你想要将刚刚我们的 hosts 内的 IP 复制到你的 /etc/hosts 这个文件去， 那么该如何编辑？我们知道在 vi 内可以使用 :r filename 来读入某个文件的内容， 不过，这样毕竟是将整个文件读入啊！如果我只是想要部分内容呢？呵呵！这个时候多文件同时编辑就很有用了。 我们可以使用 vim 后面同时接好几个文件来同时打开喔！相关的按键有：

多文件编辑的按键	
:n	编辑下一个文件
:N	编辑上一个文件
:files	列出目前这个 vim 的打开的所有文件

在过去，鸟哥想要将 A 文件内的十条消息“移动”到 B 文件去，通常要开两个 vim 窗口来复制， 偏偏每个 vim 都是独立的，因此并没有办法在 A 文件下达“ nyy ”再跑到 B 文件去“ p ”啦！ 在这种情况下最常用的方法就是通过鼠标圈选， 复制后贴上。不过这样一来还是有问题，因为鸟哥超级喜欢使用 [Tab] 按键进行编排对齐动作， 通过鼠标却会将 [Tab] 转成空白键，这样内容就不一样了！此时这个多文件编辑就派上用场了！

现在你可以做一下练习看看说！假设你要将刚刚鸟哥提供的 hosts 内的前四列 IP 数据复制到你的 /etc/hosts 文件内，那可以怎么进行呢？可以这样啊：

1. 通过“ vim hosts /etc/hosts ”指令来使用一个 vim 打开两个文件；
2. 在 vim 中先使用“ :files ”察看一下编辑的文件数据有啥？结果如下所示。 至于下图的最后一列显示的是“按下任意键”就会回到 vim 的一般指令模式中！



图9.3.6、vim 的多文件编辑中，查看同时编辑的文件数据

3. 在第一列输入“ 4yy ”复制四列；
4. 在 vim 的环境下输入“ :n ”会来到第二个编辑的文件，亦即 /etc/hosts 内；
5. 在 /etc/hosts 下按“ G ”到最后一列，再输入“ p ”贴上；
6. 按下多次的“ u ”来还原原本的文件数据；
7. 最终按下“ :q ”来离开 vim 的多文件编辑吧！

看到了吧？利用多文件编辑的功能，可以让你很快速的就将需要的数据复制到正确的文件内。 当然啰，这个功能也可以利用窗口接口来达到，那就是下面要提到的多窗口功能。

9.3.3 多窗口功能

在开始这个小节前，先来想像两个情况：

- 当我有一个文件非常的大，我查阅到后面的数据时，想要“对照”前面的数据， 是否需要使用 [ctrl]+f 与 [ctrl]+b （或 pageup, pagedown 功能键） 来跑前跑后查阅？
- 我有两个需要对照着看的文件，不想使用前一小节提到的多文件编辑功能；

在一般窗口接口下的编辑软件大多有“分区窗口”或者是“冻结窗口”的功能来将一个文件分区成多个窗口的展现， 那么 vim 能不能达到这个功能啊？可以啊！但是如何分区窗口并放入文件呢？ 很简单啊！在命令行界面输入“:sp {filename}”即可！那个 filename 可有可无， 如果想要在新窗口启动另一个文件，就加入文件名，否则仅输入 :sp 时， 出现的则是同一个文件在两个窗口间！

让我们来测试一下，你先使用“ vim /etc/man_db.conf ”打开这个文件，然后“ 1G ”去到第一列，之后输入“ :sp ” 再次的打开这个文件一次，然后再输入“ G ”，结果会变成下面这样喔：

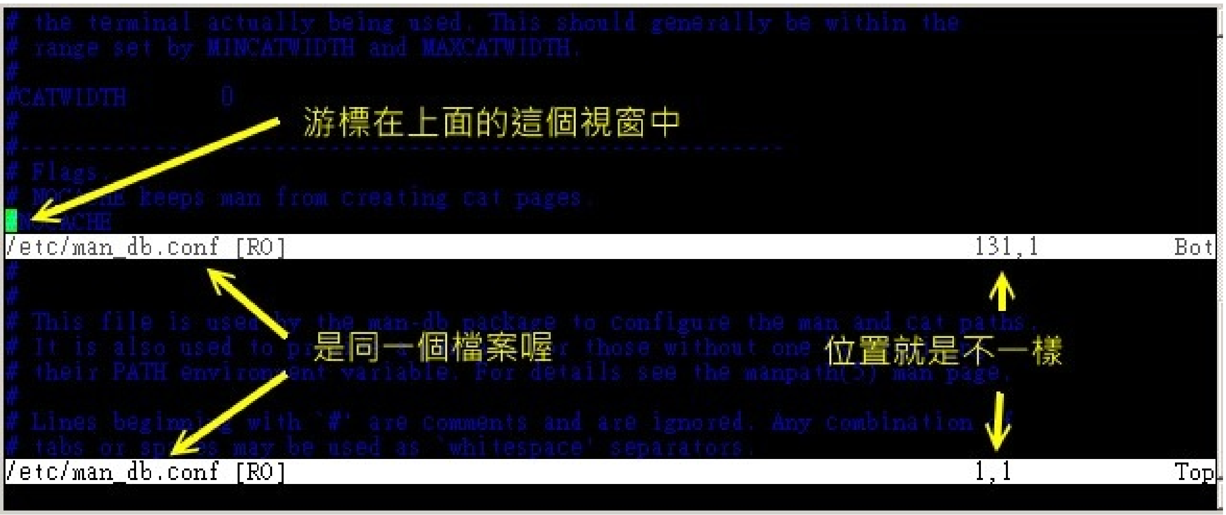


图 9.3.7、vim 的窗口分区示意图

万一你再输入“ :sp /etc/hosts ”时，就会变成下图这样喔：



图 9.3.8、vim 的窗口分区示意图

怎样？帅吧！两个文件同时在一个屏幕上面显示，你还可以利用“[ctrl]+w+↑”及“[ctrl]+w+↓” 在两个窗口之间移动呢！这样的话，复制啊、查阅啊等等的，就变的很簡单啰～ 分区窗口的相关指令功能有很多，不过你只要记得这几个就好了：

多窗口情况下的按键功能	
:sp [filename]	打开一个新窗口，如果有加 filename， 表示在新窗口打开一个新文件，否则表示两个窗口为同一个文件内容（同步显示）。
[ctrl]+w+ j [ctrl]+w+↓	按键的按法是：先按下 [ctrl] 不放， 再按下 w 后放开所有的按键，然后再按下 j （或向下方向键），则光标可移动到下方的窗口。
[ctrl]+w+ k [ctrl]+w+↑	同上，不过光标移动到上面的窗口。
[ctrl]+w+ q	其实就是 :q 结束离开啦！ 举例来说，如果我想要结束下方的窗口，那么利用 [ctrl]+w+↓ 移动到下方窗口后，按下 :q 即可离开， 也可以按下 [ctrl]+w+q 啊！

鸟哥第一次玩 vim 的分区窗口时，眞是很高兴啊！竟然有这种功能！太棒了！ ^_^

9.3.4 vim 的挑字补全功能

我们知道 bash 的环境下面可以按下 [tab] 按钮来达成指令/参数/文件名的补全功能，而我们也知道很多的程序编辑器，例如鸟哥用来在 windows 系统上面教网页设计、java script 等很好用的 notepad++ （https://notepad-plus-plus.org/） 这种类的程序编辑器，都会有 （1） 可以进行语法检验及 （2） 可以根据扩展名来挑字的功能！这两个功能对于程序设计者来说， 是很有帮助的！毕竟偶尔某些特定的关键字老是背不起来...

在语法检验方面，vim 已经使用颜色来达成了！这部份不用伤脑筋的！比较伤脑筋的应该是在挑字补全上面！就是上面谈到的可以根据语法来挑选可能的关键字， 包括程序语言的语法以及特定的语法关键字等等。既然 notepad ++ 都有支持了，没道理 vim 不支持吧？呵呵！没错！ 是有支持的～只是你可能要多背两个组合按钮就是了！

鸟哥建议可以记忆的主要 vim 补齐功能，大致有下面几个：

组合按钮	补齐的内容
[ctrl]+x -> [ctrl]+n	通过目前正在编辑的这个“文件的内容文字”作为关键字，予以补齐
[ctrl]+x -> [ctrl]+f	以当前目录内的“文件名”作为关键字，予以补齐
[ctrl]+x -> [ctrl]+o	以扩展名作为语法补充，以 vim 内置的关键字，予以补齐

在鸟哥的认知中，比较有用的是第1, 3 这两个组合键，第一个组合按键中，你可能会在同一个文件里面重复出现许多相同的关键字， 那么就能够通过这个补全的功能来处理。如果你是想要使用 vim 内置的语法检验功能来处理取得关键字的补全，那么第三个项目就很有用了。不过要注意，如果你想要使用第三个功能，就得要注意你编辑的文件的扩展名。我们下面来做个简单测试好了。

假设你想要编写网页，正要使用到 CSS 的美化功能时，突然想到有个背景的东西要处理，但是突然忘记掉背景的 CSS 关键语法，那可以使用如下的模样来处置！ 请注意，一定要使用 .html 或 .php 的扩展名，否则 vim 不会调用正确的语法检验功能喔！因此下面我们创建的文件名为 html.html 啰！

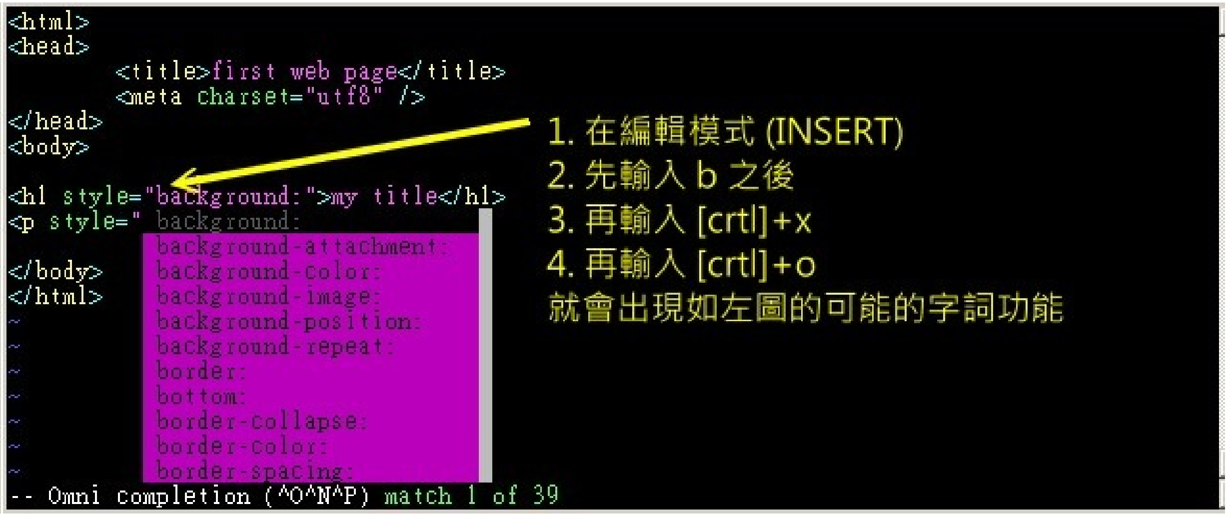


图 9.3.9、vim 的挑字补全功能

由于网页通常会支持 CSS 的语法，而 CSS 的美化语法使用的是 style 这个关键字，这个关键字后面接的就是 CSS 的元素与元素值。 若想要取得可能的元素有哪些，例如背景（background） 的语法中，想要了解有哪些跟它有关的内置元素，如上图， 直接输入 b 然后按下 [crtl]+x 再按下 [crtl]+o 就会出现如上的相关字词可以选择，此时你就能够使用上下按钮来挑选所需要的关键元素！ 这样使用上当然方便很多啊！只是要注意，一定要使用正确的扩展名，否则会无法出现任何关键字词喔！

9.3.5 vim 环境设置与记录： ~/.vimrc, ~/.viminfo

有没有发现，如果我们以 vim 软件来搜寻一个文件内部的某个字串时，这个字串会被反白，而下次我们再次以 vim 编辑这个文件时，该搜寻的字串反白情况还是存在呢！甚至于在编辑其他文件时， 如果其他文件内也存在这个字串，哇！竟然还是主动反白耶！真神奇！ 另外，当我们重复编辑同一个文件时，当第二次进入该文件时， 光标竟然就在上次离开的那一列上头呢！真是好方便啊～但是，怎么会这样呢？

这是因为我们的 vim 会主动的将你曾经做过的行为登录下来，好让你下次可以轻松的作业啊！ 那个记录动作的文件就是： ~/.viminfo ！如果你曾经使用过 vim， 那你的主文件夹应该会存在这个文件才对。这个文件是自动产生的， 你不必自行创建。而你在 vim 里头所做过的动作，就可以在这个文件内部查询到啰～ ^_^

此外，每个 distributions 对 vim 的默认环境都不太相同，举例来说，某些版本在搜寻到关键字时并不会高亮度反白， 有些版本则会主动的帮你进行缩排的行为。但这些其实都可以自行设置的，那就是 vim 的环境设置啰～ vim 的环境设置参数有很多，如果你想要知道目前的设置值，可以在一般指令模式时输入“ :set all ” 来查阅，不过.....设置项目实在太多了～所以，鸟哥在这里仅列出一些平时比较常用的一些简单的设置值， 提供给你参考啊。



Tips 所谓的缩排，就是当你按下 Enter 编辑新的一列时，光标不会在行首，而是在与上一列的第一个非空白字符处对齐！

vim 的环境设置参数	
:set nu :set nonu	就是设置与取消行号啊！
:set hlsearch :set nohlsearch	hlsearch 就是 high light search（高亮度搜寻）。 这个就是设置是否将搜寻的字串反白的设置值。默认值是 hlsearch
:set autoindent :set noautoindent	是否自动缩排？autoindent 就是自动缩排。
:set backup	是否自动储存备份文件？一般是 nobackup 的， 如果设置 backup 的话，那么当你更动任何一个文件时，则原始文件会被另存成一个文件名为 filename~ 的文件。 举例来说，我们编辑 hosts ，设置 :set backup ，那么当更动 hosts 时，在同目录下，就会产生 hosts~ 文件名的文件，记录原始的 hosts 文件内容
:set ruler	还记得我们提到的右下角的一些状态列说明吗？ 这个 ruler 就是在显示或不显示该设置值的啦！
:set showmode	这个则是，是否要显示 --INSERT-- 之类的字眼在左下角的状态列。
:set backspace=（012）	一般来说， 如果我们按下 i 进入编辑模式后，可以利用倒退键（backspace） 来删除任意字符的。 但是，某些 distribution 则不许如此。此时，我们就可以通过 backspace 来设置啰～ 当 backspace 为 2 时，就是可以删除任意值；0 或 1 时，仅可删除刚刚输入的字符， 而无法删除原本就已经存在的文字了！
:set all	显示目前所有的环境参数设置值。
:set	显示与系统默认值不同的设置参数， 一般来说就是你有自行变动过的设置参数啦！
:syntax on :syntax off	是否依据程序相关语法显示不同颜色？ 举例来说，在编辑一个纯文本文件时，如果开头是以 # 开始，那么该列就会变成蓝色。 如果你懂得写程序，那么这个 :syntax on 还会主动的帮你除错呢！但是， 如果你仅是编写纯文本，要避免颜色对你的屏幕产生的干扰，则可以取消这个设置 。
:set bg=dark :set bg=light	可用以显示不同的颜色色调，默认是“ light ”。如果你常常发现注解的字体深蓝色实在很不容易看， 那么这里可以设置为 dark 喔！试看看，会有不同的样式呢！

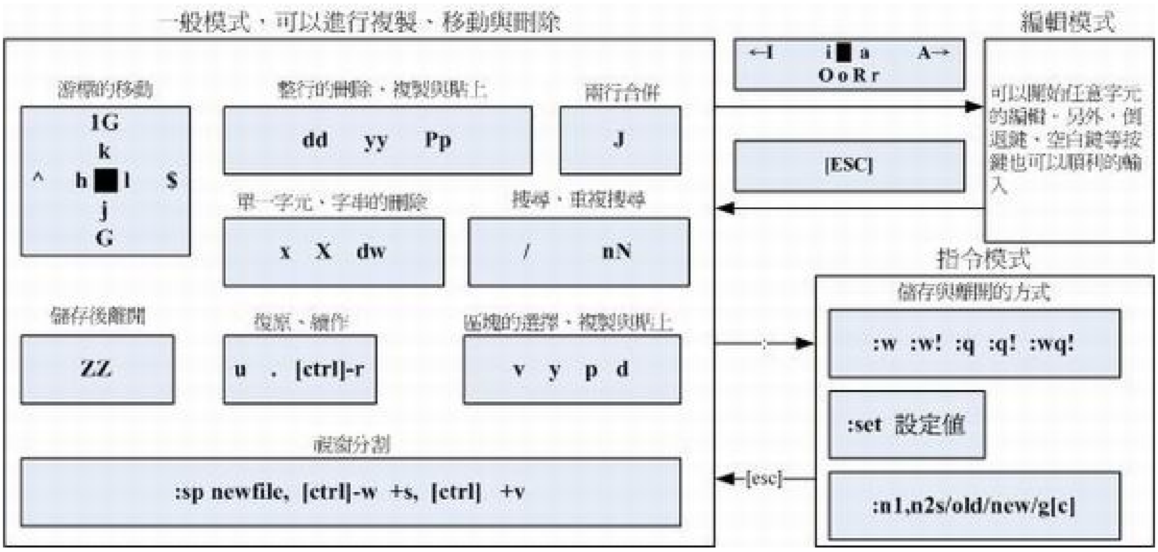
总之，这些设置值很有用处的啦！但是......我是否每次使用 vim 都要重新设置一次各个参数值？ 这不太合理吧？没错啊！所以，我们可以通过配置文件来直接规定我们习惯的 vim 操作环境呢！ 整体 vim 的设置值一般是放置在 /etc/vimrc 这个文件，不过，不建议你修改他！ 你可以修改 ~/.vimrc 这个文件 （默认不存在，请你自行手动创建！），将你所希望的设置值写入！ 举例来说，可以是这样的一个文件：

```
[dmtsai@study ~]$ vim ~/.vimrc
"这个文件的双引号 (") 是注解
set hlsearch             "高亮度反白
set backspace=2          "可随时用倒退键删除
set autoindent           "自动缩排
set ruler                "可显示最后一列的状态
set showmode             "左下角那一列的状态
set nu                   "可以在每一列的最前面显示行号啦！
set bg=dark              "显示不同的底色色调
syntax on                "进行语法检验，颜色显示。
```

在这个文件中，使用“ set hlsearch ”或“ :set hlsearch ”，亦即最前面有没有冒号“ : ”效果都是一样的！ 至于双引号则是注解符号！不要用错注解符号，否则每次使用 vim 时都会发生警告讯息喔！ 创建好这个文件后，当你下次重新以 vim 编辑某个文件时，该文件的默认环境设置就是上头写的啰～ 这样，是否很方便你的操作啊！多多利用 vim 的环境设置功能呢！^_^

9.3.6 vim 常用指令示意图

为了方便大家查询在不同的模式下可以使用的 vim 指令，鸟哥查询了一些 vim 与 Linux 教育训练手册， 发现下面这张图非常值得大家参考！可以更快速有效的查询到需要的功能喔！看看吧！



图

9.3.10、vim 常用指令示意图

9.4 其他 vim 使用注意事项

vim 其实不是那么好学，虽然他的功能确实非常强大！所以下面我们还有一些需要注意的地方要来跟大家分享喔！

9.4.1 中文编码的问题

很多朋友常常哀嚎，说他们的 vim 里面怎么无法显示正常的中文啊？其实这很有可能是因为编码的问题！ 因为中文编码有 big5 与 utf8 两种，如果你的文件是使用 big5 编码制作的，但在 vim 的终端接口中你使用的是万国码（utf8）， 由于编码的不同，你的中文文件内容当然就是一堆乱码了！怎么办？这时你得要考虑许多东西啦！有这些：

1. 你的 Linux 系统默认支持的语系数据：这与 /etc/locale.conf 有关；
2. 你的终端接口 （bash） 的语系： 这与 LANG, LC_ALL 这几个变量有关；
3. 你的文件原本的编码；
4. 打开终端机的软件，例如在 GNOME 下面的窗口接口。

事实上最重要的是上头的第三与第四点，只要这两点的编码一致，你就能够正确的看到与编辑你的中文文件。 否则就会看到一堆乱码啦！

一般来说，中文编码使用 big5 时，在写入某些数据库系统中，在“许、盖、功”这些字体上面会发生错误！ 所以近期以来大多希望大家能够使用万国码 utf8 来进行中文编码！但是在中文 Windows 上的软件常常默认使用 big5 的编码 （不一定是 windows 系统的问题，有时候是某些中文软件的默认值之故）， 包括鸟哥由于沿用以前的文件数据文件，也大多使用 big5 的编码。此时就得要注意上述的这些咚咚啰。

在 Linux 本机前的 tty1~tty6 原本默认就不支持中文编码，所以不用考虑这个问题！因为你一定会看到乱码！呵呵！ 现在鸟哥假设俺的文件文件内编码为 big5 时，而且我的环境是使用 Linux 的 GNOME ，启动的终端接口为 GNOME-terminal 软件， 那鸟哥通常是这样来修正语系编码的行为：

```
[dmtsai@study ~]$ LANG=zh_TW.big5
[dmtsai@study ~]$ export LC_ALL=zh_TW.big5
```

然后在终端接口工具列的“终端机”-->“设置字符编码” -->“中文 （正体） （BIG5）”项目点选一下， 如果一切都没有问题了，再用 vim 去打开那个 big5 编码的文件，就没有问题了！以上！报告完毕！

9.4.2 DOS 与 Linux 的断行字符

我们在第六章里面谈到 cat 这个指令时，曾经提到过 DOS 与 Linux 断行字符的不同。 而我们也可以利用 cat -A 来观察以 DOS （Windows 系统） 创建的文件的特殊格式， 也可以发现在 DOS 使用的断行字符为 ^M$ ，我们称为 CR 与 LF 两个符号。 而在 Linux 下面，则是仅有 LF （$） 这个断行符号。这个断行符号对于 Linux 的影响很大喔！ 为什么呢？

我们说过，在 Linux 下面的指令在开始执行时，他的判断依据是 “Enter”，而 Linux 的 Enter 为 LF 符号， 不过，由于 DOS 的断行符号是 CRLF ，也就是多了一个 ^M 的符号出来， 在这样的情况下，如果是一个 shell script 的程序文件，呵呵～将可能造成“程序无法执行”的状态～ 因为他会误判程序所下达的指令内容啊！这很伤脑筋吧！

那怎么办啊？很简单啊，将格式转换成为 Linux 即可啊！“废话”，这当然大家都知道，但是，要以 vi 进入该文件，然后一个一个删除每一列的 CR 吗？当然没有这么没人性啦！ 我们可以通过简单的指令来进行格式的转换啊！

不过，由于我们要操作的指令默认并没有安装，鸟哥也无法预期你有没有网络，因此假设你没有网络的状况下， 请拿出你的原版光盘，放到光驱里头去，然后使用下面的方式来安装我们所需要的这个软件喔！

```
[dmtsai@study ~]$ su -   # 安装软件一定要是 root 的权限才行！
[root@study ~]# mount /dev/sr0 /mnt
[root@study ~]# rpm -ivh /mnt/Packages/dos2unix-*
warning: /mnt/Packages/dos2unix-6.0.3-4.el7.x86_64.rpm: Header V3 RSA/SHA256 ....
Preparing...                          ################################# [100%]
Updating / installing...
   1:dos2unix-6.0.3-4.el7             ################################# [100%]
[root@study ~]# umount /mnt
[root@study ~]# exit
```

那就开始来玩一玩这个字符转换吧！

```
[dmtsai@study ~]$ dos2unix [-kn] file [newfile]
[dmtsai@study ~]$ unix2dos [-kn] file [newfile]
选项与参数：
-k  ：保留该文件原本的 mtime 时间格式 （不更新文件上次内容经过修订的时间）
-n  ：保留原本的旧文件，将转换后的内容输出到新文件，如： dos2unix -n old new

范例一：将 /etc/man_db.conf 重新复制到 /tmp/vitest/ 下面，并将其修改成为 dos 断行
[dmtsai@study ~]# cd /tmp/vitest
[dmtsai@study vitest]$ cp -a /etc/man_db.conf .
[dmtsai@study vitest]$ ll man_db.conf
-rw-r--r--. 1 root root 5171 Jun 10  2014 man_db.conf
[dmtsai@study vitest]$ unix2dos -k man_db.conf
unix2dos: converting file man_db.conf to DOS format ...
# 屏幕会显示上述的讯息，说明断行转为 DOS 格式了！
[dmtsai@study vitest]$ ll man_db.conf
-rw-r--r--. 1 dmtsai dmtsai 5302 Jun 10  2014 man_db.conf
# 断行字符多了 ^M ，所以容量增加了！

范例二：将上述的 man_db.conf 转成 Linux 断行字符，并保留旧文件，新文件放于 man_db.conf.linux
[dmtsai@study vitest]$ dos2unix -k -n man_db.conf man_db.conf.linux
dos2unix: converting file man_db.conf to file man_db.conf.linux in Unix format ...
[dmtsai@study vitest]$ ll man_db.conf*
-rw-r--r--. 1 dmtsai dmtsai 5302 Jun 10  2014 man_db.conf
-rw-r--r--. 1 dmtsai dmtsai 5171 Jun 10  2014 man_db.conf.linux
[dmtsai@study vitest]$ file man_db.conf*
man_db.conf:       ASCII text, with CRLF line terminators  # 很清楚说明是 CRLF 断行！
man_db.conf.linux: ASCII text
```

因为断行字符以及 DOS 与 Linux 操作系统下面一些字符的定义不同，因此， 不建议你在 Windows 系统当中将文件编辑好之后，才上传到 Linux 系统，会容易发生错误问题。 而且，如果你在不同的系统之间复制一些纯文本时，千万记得要使用 unix2dos 或 dos2unix 来转换一下断行格式啊！

9.4.3 语系编码转换

很多朋友都会有的问题，就是想要将语系编码进行转换啦！举例来说，想要将 big5 编码转成 utf8 。 这个时候怎么办？难不成要每个文件打开会转存成 utf8 吗？不需要这样做啦！使用 iconv 这个指令即可！ 鸟哥将之前的 vi 章节做成 big5 编码的文件，你可以照下面的链接来下载先：

- http://linux.vbird.org/linux_basic/0310vi/vi.big5

在终端机的环境下你可以使用“ wget 网址”来下载上述的文件喔！鸟哥将他下载在 /tmp/vitest 目录下。 接下来让我们来使用 iconv 这个指令来玩一玩编码转换吧！

```
[dmtsai@study ~]$ iconv --list
[dmtsai@study ~]$ iconv -f 原本编码 -t 新编码 filename [-o newfile]
选项与参数：
--list ：列出 iconv 支持的语系数据
-f     ：from ，亦即来源之意，后接原本的编码格式；
-t     ：to ，亦即后来的新编码要是什么格式；
-o file：如果要保留原本的文件，那么使用 -o 新文件名，可以创建新编码文件。

范例一：将 /tmp/vitest/vi.big5 转成 utf8 编码吧！
[dmtsai@study ~]$ cd /tmp/vitest
[dmtsai@study vitest]$ iconv -f big5 -t utf8 vi.big5 -o vi.utf8
[dmtsai@study vitest]$ file vi*
vi.big5: ISO-8859 text, with CRLF line terminators
vi.utf8: UTF-8 Unicode text, with CRLF line terminators
# 是吧！有明显的不同吧！ ^_^
```

这指令支持的语系非常之多，除了正体中文的 big5, utf8 编码之外，也支持简体中文的 gb2312 ， 所以对岸的朋友可以简单的将鸟站的网页数据下载后，利用这个指令来转成简体，就能够轻松的读取文件数据啰！ 不过，不要将转成简体的文件又上传成为您自己的网页啊！ 这明明是鸟哥写的不是吗？ ^_^

不过如果是要将正体中文的 utf8 转成简体中文的 utf8 编码时，那就得费些功夫了！ 举例来说，如果要将刚刚那个 vi.utf8 转成简体的 utf8 时，可以这样做：

```
[dmtsai@study vitest]$ iconv -f utf8 -t big5 vi.utf8 &#124; \
&gt; iconv -f big5 -t gb2312 &#124; iconv -f gb2312 -t utf8 -o vi.gb.utf8
```

9.5 重点回顾

- Linux 下面的配置文件多为文本文件，故使用 vim 即可进行设置编辑；
- vim 可视为程序编辑器，可用以编辑 shell script, 配置文件等，避免打错字；
- vi 为所有 unix like 的操作系统都会存在的编辑器，且执行速度快速；
- vi 有三种模式，一般指令模式可变换到编辑与命令行界面，但编辑模式与命令行界面不能互换；
- 常用的按键有i, [Esc], :wq 等；
- vi 的画面大略可分为两部份，（1）上半部的本文与（2）最后一行的状态+命令行界面；
- 数字是有意义的，用来说明重复进行几次动作的意思，如 5yy 为复制 5 列之意；
- 光标的移动中，大写的 G 经常使用，尤其是 1G, G 移动到文章的头/尾功能！
- vi 的取代功能也很棒！ :n1,n2s/old/new/g 要特别注意学习起来；
- 小数点“ . ”为重复进行前一次动作，也是经常使用的按键功能！
- 进入编辑模式几乎只要记住： i, o, R 三个按钮即可！尤其是新增一列的 o 与取代的 R
- vim 会主动的创建 swap 暂存盘，所以不要随意断线！
- 如果在文章内有对齐的区块，可以使用 [ctrl]-v 进行复制/贴上/删除的行为
- 使用 :sp 功能可以分区窗口
- 若使用 vim 来撰写网页，若需要 CSS 元素数据，可通过 [crtl]+x, [crtl]+o 这两个连续组合按键来取得关键字
- vim 的环境设置可以写入在 ~/.vimrc 文件中；
- 可以使用 iconv 进行文件语系编码的转换
- 使用 dos2unix 及 unix2dos 可以变更文件每一列的行尾断行字符。

9.6 本章练习

（要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看） 实作题部分：

- 在第七章的情境仿真题二的第五点，编写 /etc/fstab 时，当时使用 nano 这个指令， 请尝试使用 vim 去编辑 /etc/fstab ，并且将第七章新增的那一列的 defatuls 改成 default ，会出现什么状态？ 离开前请务必要修订成原本正确的信息。此外，如果将该列注解 （最前面加 #），你会发现字体颜色也有变化喔！
- 尝试在你的系统中，你惯常使用的那个帐号的主文件夹下，将本章介绍的 vimrc 内容进行一些常用设置，包括：
    - 设置搜寻高亮度反白
    - 设置语法检验启动
    - 设置默认启动行号显示
    - 设置有两行状态列 （一行状态+一行命令行） :set laststatus=2

简答题部分：

- 我用 vi 打开某个文件后，要在第 34 列向右移动 15 个字符，应该在一般指令模式中下达什么指令？（1）先按下 34G 到第 34 列；（2）再按下 [15 + 向右键]，或 [15l] 亦可！
- 在 vi 打开的文件中，如何去到该文件的页首或页尾？去页首按下 1G 或 gg ；去页尾按下 G 即可
- 在 vi 打开的文件中，如何在光标所在列中，移动到行头及行尾？移动到行头，按 0 ，移动到行尾按 $ 即可！
- vi 的一般指令模式情况下，按下“ r ”有什么功能？取代光标所在的那个字符
- 在 vi 的环境中，如何将目前正在编辑的文件另存新文件名为 newfilename？:w newfilename
- 在 linux 下面最常使用的文书编辑器为 vi ，请问如何进入编辑模式？在一般指令模式下面输入： i, I, a, A 为在本列当中输入新字符；（出现 –Insert- ） 在一般指令模式当中输入： o, O 为在一个新的一列输入新字符； 在一般指令模式当中输入： r, R 为取代字符！（左下角出现 –Replace-）
- 在 vi 软件中，如何由编辑模式跳回一般指令模式？可以按下[Esc]
- 在 vi 环境中，若上下左右键无法使用时，请问如何在一般指令模式移动光标？[h, j, k, l] 分别代表[左、下、上、右]
- 在 vi 的一般指令模式中，如何删除一列、n列；如何删除一个字符？分别为 dd, ndd, x 或 X （dG 及 d1G 分别表示删除到页首及页尾）
- 在 vi 的一般指令模式中，如何复制一列、n列并加以贴上？分别为 yy, nyy, p 或 P

- 在 vi 的一般指令模式中如何搜寻 string 这个字串？?string （往前搜寻） /string （往后搜寻）
- 在 vi 的一般指令模式中，如何取代 word1 成为 word2，而若需要使用者确认机制，又该如何？:1,$s/word1/word2/g 或 :1,$s/word1/word2/gc （需要使用者确认）
- 在 vi 目前的编辑文件中，在一般指令模式下，如何读取一个文件 filename 进来目前这个文件？:r filename
- 在 vi 的一般指令模式中，如何存盘、离开、存盘后离开、强制存盘后离开？:w； :q： :wq； :wq!
- 在 vi 下面作了很多的编辑动作之后，却想还原成原来的文件内容，应该怎么进行？直接按下 :e! 即可恢复成文件的原始状态！
- 我在 vi 这个程序当中，不想离开 vi ，但是想执行 ls /home 这个指令，vi 有什么额外的功能可以达到这个目的：事实上，可以使用[:! ls /home]不过，如果你学过后面的章节之后，你会发现，执行[ctrl + z]亦可暂时退出 vi 让你在命令行界面当中执行指令喔！

9.7 参考资料与延伸阅读

- [1]常见文书编辑器专案计划链接：
    - emacs: http://www.gnu.org/software/emacs/
    - pico: https://en.wikipedia.org/wiki/Pico_（text_editor）
    - nano: http://sourceforge.net/projects/nano/
    - joe: http://sourceforge.net/projects/joe-editor/
    - vim: http://www.vim.org
    - 常见文书编辑器比较：http://encyclopedia.thefreedictionary.com/List+of+text+editors
    - 维基百科的文书编辑器比较：http://en.wikipedia.org/wiki/Comparison_of_text_editors
- 维基百科：ASCII 的代码与图示对应表：http://zh.wikipedia.org/wiki/ASCII
- 关于 vim 是什么的“中文”说明：http://www.vim.org/6k/features.zh.txt。
- vim 补齐功能介绍：http://www.openfoundry.org/en/tech-column/2215

2002/04/05：第一次完成 2003/02/07：重新编排与加入 FAQ 2003/02/25：新加入本章节与 LPI 的相关性说明！ 2005/07/28：将旧文章移动到 这里 。 2005/08/01：加入果正凡文章的参考，还有查阅 vim 官方网站的数据！ 2008/12/18：将原本针对 FC4 版本的文章移动到 此处 2009/01/13：这么简单的一篇改写，竟改了一个月！原因只是期末考将近太忙了～ 2009/08/20：加入实作题，编辑简答题，加入 vim 指令示意图等

第十章、认识与学习BASH

最近更新日期：20//

在 Linux 的环境下，如果你不懂 bash 是什么，那么其他的东西就不用学了！因为前面几章我们使用终端机下达指令的方式， 就是通过 bash 的环境来处理的喔！所以说，他很重要吧！ bash 的东西非常的多，包括变量的设置与使用、 bash 操作环境的创建、数据流重导向的功能，还有那好用的管线命令！好好清一清脑门，准备用功去啰～ ^_^ 这个章节几乎是所有命令行界面 （command line） 与未来主机维护与管理的重要基础，一定要好好仔细的阅读喔！

10.1 认识 BASH 这个 Shell

我们在第一章 Linux 是什么当中提到了： 管理整个计算机硬件的其实是操作系统的核心（kernel），这个核心是需要被保护的！ 所以我们一般使用者就只能通过 shell 来跟核心沟通，以让核心达到我们所想要达到的工作。 那么系统有多少 shell 可用呢？为什么我们要使用 bash 啊？下面分别来谈一谈喔！

10.1.1 硬件、核心与 Shell

这应该是个蛮有趣的话题："什么是 Shell "？相信只要摸过计算机，对于操作系统 （不论是 Linux 、 Unix 或者是 Windows） 有点概念的朋友们大多听过这个名词，因为只要有"操作系统"那么就离不开 Shell 这个东西。不过，在讨论 Shell 之前，我们先来了解一下计算机的运行状况吧！ 举个例子来说：当你要计算机传输出来"音乐"的时候，你的计算机需要什么东西呢？

1. 硬件：当然就是需要你的硬件有"声卡芯片"这个配备，否则怎么会有声音；
2. 核心管理：操作系统的核心可以支持这个芯片组，当然还需要提供芯片的驱动程序啰；
3. 应用程序：需要使用者 （就是你） 输入发生声音的指令啰！

这就是基本的一个输出声音所需要的步骤！也就是说，你必须要"输入"一个指令之后， "硬件"才会通过你下达的指令来工作！那么硬件如何知道你下达的指令呢？那就是 kernel （核心） 的控制工作了！也就是说，我们必须要通过" Shell "将我们输入的指令与 Kernel 沟通，好让 Kernel 可以控制硬件来正确无误的工作！ 基本上，我们可以通过下面这张图来说明一下：

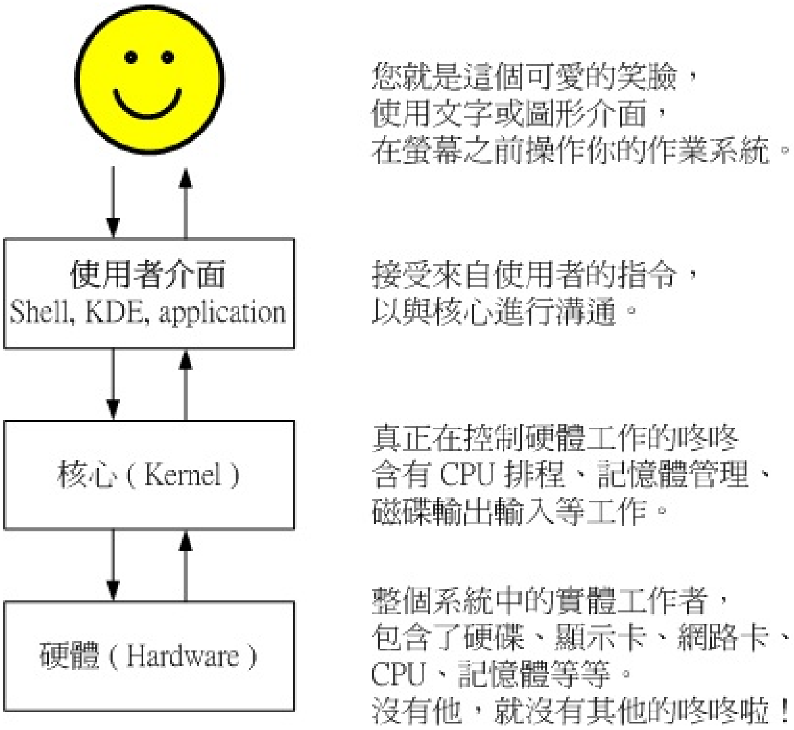


图10.1.1、硬件、核心与使用者的相关性图示

我们在第零章内的操作系统小节曾经提到过， 操作系统其实是一组软件，由于这组软件在控制整个硬件与管理系统的活动监测， 如果这组软件能被使用者随意的操作，若使用者应用不当，将会使得整个系统崩溃！因为操作系统管理的就是整个硬件功能嘛！ 所以当然不能够随便被一些没有管理能力的终端用户随意使用啰！

但是我们总是需要让使用者操作系统的，所以就有了在操作系统上面发展的应用程序啦！使用者可以通过应用程序来指挥核心， 让核心达成我们所需要的硬件任务！如果考虑如第零章所提供的操作系统图示（图0.4.2）， 我们可以发现应用程序其实是在最外层，就如同鸡蛋的外壳一样，因此这个咚咚也就被称呼为壳程序 （shell） 啰！

其实壳程序的功能只是提供使用者操作系统的一个接口，因此这个壳程序需要可以调用其他软件才好。 我们在第四章到第九章提到过很多指令，包括 man, chmod, chown, vi, fdisk, mkfs 等等指令，这些指令都是独立的应用程序， 但是我们可以通过壳程序 （就是命令行界面） 来操作这些应用程序，让这些应用程序调用核心来运行所需的工作哩！ 这样对于壳程序是否有了一定的概念了？



Tips 也就是说，只要能够操作应用程序的接口都能够称为壳程序。狭义的壳程序指的是命令行方面的软件，包括本章要介绍的 bash 等。 广义的壳程序则包括图形接口的软件！因为图形接口其实也能够操作各种应用程序来调用核心工作啊！ 不过在本章中，我们主要还是在使用 bash 啦！

10.1.2 为何要学命令行的 shell？

命令行的 shell 是很不好学的，但是学了之后好处多多！所以， 在这里鸟哥要先对您进行一些心理建设，先来了解一下为啥学习 shell 是有好处的，这样你才会有信心继续玩下去 ^_^

- 命令行的 shell：大家都一样！

鸟哥常常听到这个问题："我干嘛要学习 shell 呢？ 不是已经有很多的工具可以提供我设置我的主机了？我为何要花这么多时间去学指令呢？不是以 X Window 按一按几个按钮就可以搞定了吗？"唉～还是得一再地强调， X Window 还有 Web 接口的设置工具例如 Webmin [1] 是真的好用的家伙， 他真的可以帮助我们很简易的设置好我们的主机，甚至是一些很进阶的设置都可以帮我们搞定。

但是鸟哥在前面的章节里面也已经提到过相当多次了， X Window 与 web 接口的工具，他的接口虽然好友，功能虽然强大， 但毕竟他是将所有利用到的软件都整合在一起的一组应用程序而已， 并非是一个完整的套件，所以某些时候当你升级或者是使用其他套件管理模块 （例如 tarball 而非 rpm 文件等等） 时，就会造成设置的困扰了。甚至不同的 distribution 所设计的 X window 接口也都不相同，这样也造成学习方面的困扰。

命令行的 shell 就不同了！几乎各家 distributions 使用的 bash 都是一样的！如此一来， 你就能够轻轻松松的转换不同的 distributions ，就像武侠小说里面提到的"一法通、万法通！"

- 远端管理：命令行就是比较快！

此外，Linux 的管理常常需要通过远端连线，而连线时命令行的传输速度一定比较快， 而且，较不容易出现断线或者是信息外流的问题，因此，shell 真的是得学习的一项工具。而且，他可以让您更深入 Linux ，更了解他，而不是只会按一按鼠标而已！所谓"天助自助者！"多摸一点文字模式的东西，会让你与 Linux 更亲近呢！

- Linux 的任督二脉： shell 是也！

有些朋友也很可爱，常会说："我学这么多干什么？ 又不常用，也用不到！"嘿嘿！有没有听过"书到用时方恨少？" 当你的主机一切安然无恙的时候，您当然会觉得好像学这么多的东西一点帮助也没有呀！ 万一，某一天真的不幸给他中标了，您该如何是好？是直接重新安装？还是先追踪入侵来源后进行漏洞的修补？或者是干脆就关站好了？这当然涉及很多的考虑，但就以鸟哥的观点来看，多学一点总是好的，尤其我们可以有备而无患嘛！甚至学的不精也没有关系，了解概念也就 OK 啦！毕竟没有人要您一定要背这么多的内容啦！了解概念就很了不起了！

此外，如果你真的有心想要将您的主机管理的好，那么良好的 shell 程序编写是一定需要的啦！就鸟哥自己来说，鸟哥管理的主机虽然还不算多， 只有区区不到十部，但是如果每部主机都要花上几十分钟来查阅他的登录文件信息以及相关的讯息， 那么鸟哥可能会疯掉！基本上，也太没有效率了！这个时候，如果能够借由 shell 提供的数据流重导向以及管线命令，呵呵！ 那么鸟哥分析登录信息只要花费不到十分钟就可以看完所有的主机之重要信息了！相当的好用呢！

由于学习 shell 的好处真的是多多啦！所以，如果你是个系统管理员，或者有心想要管理系统的话，那么 shell 与 shell scripts 这个东西真的有必要看一看！因为他就像“打通任督二脉，任何武功都能随你应用”的说！

10.1.3 系统的合法 shell 与 /etc/shells 功能

知道什么是 Shell 之后，那么我们来了解一下 Linux 使用的是哪一个 shell 呢？什么！哪一个？难道说 shell 不就是“一个 shell 吗？”哈哈！那可不！由于早年的 Unix 年代，发展者众，所以由于 shell 依据发展者的不同就有许多的版本，例如常听到的 Bourne SHell （sh） 、在 Sun 里头默认的 C SHell、 商业上常用的 K SHell、, 还有 TCSH 等等，每一种 Shell 都各有其特点。至于 Linux 使用的这一种版本就称为“ Bourne Again SHell （简称 bash） ”，这个 Shell 是 Bourne Shell 的增强版本，也是基准于 GNU 的架构下发展出来的呦！

在介绍 shell 的优点之前，先来说一说 shell 的简单历史吧[2]：第一个流行的 shell 是由 Steven Bourne 发展出来的，为了纪念他所以就称为 Bourne shell ，或直接简称为 sh ！而后来另一个广为流传的 shell 是由柏克莱大学的 Bill Joy 设计依附于 BSD 版的 Unix 系统中的 shell ，这个 shell 的语法有点类似 C 语言，所以才得名为 C shell ，简称为 csh ！由于在学术界 Sun 主机势力相当的庞大，而 Sun 主要是 BSD 的分支之一，所以 C shell 也是另一个很重要而且流传很广的 shell 之一 。



Tips 由于 Linux 为 C 程序语言撰写的，很多程序设计师使用 C 来开发软件，因此 C shell 相对的就很热门了。 另外，还记得我们在第一章、Linux 是什么提到的吧？ Sun 公司的创始人就是 Bill Joy，而 BSD 最早就是 Bill Joy 发展出来的啊。

那么目前我们的 Linux （以 CentOS 7.x 为例） 有多少我们可以使用的 shells 呢？ 你可以检查一下 /etc/shells 这个文件，至少就有下面这几个可以用的 shells （鸟哥省略了重复的 shell 了！包括 /bin/sh 等于 /usr/bin/sh 啰！） ：

- /bin/sh （已经被 /bin/bash 所取代）
- /bin/bash （就是 Linux 默认的 shell）
- /bin/tcsh （整合 C Shell ，提供更多的功能）
- /bin/csh （已经被 /bin/tcsh 所取代）

虽然各家 shell 的功能都差不多，但是在某些语法的下达方面则有所不同，因此建议你还是得要选择某一种 shell 来熟悉一下较佳。 Linux 默认就是使用 bash ，所以最初你只要学会 bash 就非常了不起了！ ^_^！ 另外，咦！为什么我们系统上合法的 shell 要写入 /etc/shells 这个文件啊？ 这是因为系统某些服务在运行过程中，会去检查使用者能够使用的 shells ，而这些 shell 的查询就是借由 /etc/shells 这个文件啰！

举例来说，某些 FTP 网站会去检查使用者的可用 shell ，而如果你不想要让这些使用者使用 FTP 以外的主机资源时，可能会给予该使用者一些怪怪的 shell，让使用者无法以其他服务登陆主机。 这个时候，你就得将那些怪怪的 shell 写到 /etc/shells 当中了。举例来说，我们的 CentOS 7.x 的 /etc/shells 里头就有个 /sbin/nologin 文件的存在，这个就是我们说的怪怪的 shell 啰～

那么，再想一想，我这个使用者什么时候可以取得 shell 来工作呢？还有， 我这个使用者默认会取得哪一个 shell 啊？还记得我们在第四章的在终端接口登陆linux小节当中提到的登陆动作吧？ 当我登陆的时候，系统就会给我一个 shell 让我来工作了。 而这个登陆取得的 shell 就记录在 /etc/passwd 这个文件内！这个文件的内容是啥？

```
[dmtsai@study ~]$ cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
.....（下面省略）.....
```

如上所示，在每一行的最后一个数据，就是你登陆后可以取得的默认的 shell 啦！那你也会看到， root 是 /bin/bash ，不过，系统帐号 bin 与 daemon 等等，就使用那个怪怪的 /sbin/nologin 啰～关于使用者这部分的内容，我们留在第十三章的帐号管理时提供更多的说明。

10.1.4 Bash shell 的功能

既然 /bin/bash 是 Linux 默认的 shell ，那么总是得了解一下这个玩意儿吧！bash 是 GNU 计划中重要的工具软件之一，目前也是 Linux distributions 的标准 shell 。 bash 主要相容于 sh ，并且依据一些使用者需求而加强的 shell 版本。不论你使用的是那个 distribution ，你都难逃需要学习 bash 的宿命啦！那么这个 shell 有什么好处，干嘛 Linux 要使用他作为默认的 shell 呢？ bash 主要的优点有下面几个：

- 命令编修能力 （history）：

bash 的功能里头，鸟哥个人认为相当棒的一个就是“他能记忆使用过的指令！” 这功能真的相当的棒！因为我只要在命令行按“上下键”就可以找到前/后一个输入的指令！而在很多 distribution 里头，默认的指令记忆功能可以到达 1000 个！也就是说，你曾经下达过的指令几乎都被记录下来了。

这么多的指令记录在哪里呢？在你的主文件夹内的 .bash_history 啦！ 不过，需要留意的是，~/.bash_history 记录的是前一次登陆以前所执行过的指令， 而至于这一次登陆所执行的指令都被暂存在内存中，当你成功的登出系统后，该指令记忆才会记录到 .bash_history 当中！

这有什么优点呢？最大的好处就是可以“查询曾经做过的举动！” 如此可以知道你的执行步骤，那么就可以追踪你曾下达过的指令，以作为除错的重要流程！ 但如此一来也有个烦恼，就是如果被骇客入侵了，那么他只要翻你曾经执行过的指令， 刚好你的指令又跟系统有关

（例如直接输入 MySQL 的密码在命令行上面），那你的服务器可就伤脑筋了！ 到底记录指令的数目越多还是越少越好？这部份是见仁见智啦，没有一定的答案的。

- 命令与文件补全功能：（[tab] 按键的好处）

还记得我们在第四章内的重要的几个热键小节当中提到的 [tab] 这个按键吗？这个按键的功能就是在 bash 里头才有的啦！常常在 bash 环境中使用 [tab] 是个很棒的习惯喔！因为至少可以让你 1）少打很多字；2）确定输入的数据是正确的！ 使用 [tab] 按键的时机依据 [tab] 接在指令后或参数后而有所不同。我们再复习一次：

- [Tab] 接在一串指令的第一个字的后面，则为命令补全；
- [Tab] 接在一串指令的第二个字以后时，则为“文件补齐”！
- 若安装 bash-completion 软件，则在某些指令后面使用 [tab] 按键时，可以进行“选项/参数的补齐”功能！

所以说，如果我想要知道我的环境当中所有以 c 为开头的指令呢？就按下“ c[tab][tab] ”就好啦！ ^_^！ 是的！真的是很方便的功能，所以，有事没事，在 bash shell 下面，多按几次 [tab] 是一个不错的习惯啦！

- 命令别名设置功能：（alias）

假如我需要知道这个目录下面的所有文件 （包含隐藏文件） 及所有的文件属性，那么我就必须要下达“ ls -al ”这样的指令串，唉！真麻烦，有没有更快的取代方式？呵呵！就使用命令别名呀！例如鸟哥最喜欢直接以 lm 这个自订的命令来取代上面的命令，也就是说， lm 会等于 ls -al 这样的一个功能，嘿！那么要如何作呢？就使用 alias 即可！你可以在命令行输入 alias 就可以知道目前的命令别名有哪些了！也可以直接下达命令来设置别名呦：

- alias lm='ls -al'

- 工作控制、前景背景控制：（job control, foreground, background）

这部分我们在第十六章 Linux 程序控制中再提及！ 使用前、背景的控制可以让工作进行的更为顺利！至于工作控制（jobs）的用途则更广， 可以让我们随时将工作丢到背景中执行！而不怕不小心使用了 [Ctrl] + c 来停掉该程序！真是好样的！此外，也可以在单一登陆的环境中，达到多任务的目的呢！

- 程序化脚本：（shell scripts）

在 DOS 年代还记得将一堆指令写在一起的所谓的“批处理文件”吧？在 Linux 下面的 shell scripts 则发挥更为强大的功能，可以将你平时管理系统常需要下达的连续指令写成一个文件， 该文件并且可以通过对谈互动式的方式来进行主机的侦测工作！也可以借由 shell 提供的环境变量及相关指令来进行设计，哇！整个设计下来几乎就是一个小型的程序语言了！该 scripts 的功能真的是超乎鸟哥的想像之外！以前在 DOS 下面需要程序语言才能写的东西，在 Linux 下面使用简单的 shell scripts 就可以帮你达成了！真的厉害！这部分我们在第十二章再来谈！

- 万用字符：（Wildcard）

除了完整的字串之外， bash 还支持许多的万用字符来帮助使用者查询与指令下达。 举例来说，想要知道 /usr/bin 下面有多少以 X 为开头的文件吗？使用：“ ls -l /usr/bin/X* ”就能够知道啰～此外，还有其他可供利用的万用字符， 这些都能够加快使用者的操作呢！

总之，bash 这么好！不学吗？怎么可能！来学吧！ ^_^

10.1.5 查询指令是否为 Bash shell 的内置命令： type

我们在第四章提到关于 Linux 的线上说明文档部分，也就是 man page 的内容，那么 bash 有没有什么说明文档啊？开玩笑～ 这么棒的东西怎么可能没有说明文档！请你在 shell 的环境下，直接输入 man bash 瞧一瞧， 嘿嘿！不是盖的吧！让你看个几天几夜也无法看完的 bash 说明文档，可是很详尽的数据啊！ ^_^

不过，在这个 bash 的 man page 当中，不知道你是否有察觉到，咦！ 怎么这个说明文档里面有其他的文件说明啊？举例来说，那个 cd 指令的说明就在这个 man page 内？ 然后我直接输入 man cd 时，怎么出现的画面中，最上方竟然出现一堆指令的介绍？这是怎么回事？ 为了方便 shell 的操作，其实 bash 已经“内置”了很多指令了，例如上面提到的 cd ， 还有例如 umask 等等的指令，都是内置在 bash 当中的呢！

那我怎么知道这个指令是来自于外部指令（指的是其他非 bash 所提供的指令） 或是内置在 bash 当中的呢？ 嘿嘿！利用 type 这个指令来观察即可！举例来说：

```
[dmtsai@study ~]$ type [-tpa] name
选项与参数：
    ：不加任何选项与参数时，type 会显示出 name 是外部指令还是 bash 内置指令
-t  ：当加入 -t 参数时，type 会将 name 以下面这些字眼显示出他的意义：
      file    ：表示为外部指令；
      alias   ：表示该指令为命令别名所设置的名称；
      builtin ：表示该指令为 bash 内置的指令功能；
-p  ：如果后面接的 name 为外部指令时，才会显示完整文件名；
-a  ：会由 PATH 变量定义的路径中，将所有含 name 的指令都列出来，包含 alias

范例一：查询一下 ls 这个指令是否为 bash 内置？
[dmtsai@study ~]$ type ls
ls is aliased to `ls --color=auto' &lt;==未加任何参数，列出 ls 的最主要使用情况
[dmtsai@study ~]$ type -t ls
alias                              &lt;==仅列出 ls 执行时的依据
[dmtsai@study ~]$ type -a ls
ls is aliased to `ls --color=auto' &lt;==最先使用 aliase
ls is /usr/bin/ls                  &lt;==还有找到外部指令在 /bin/ls

范例二：那么 cd 呢？
[dmtsai@study ~]$ type cd
cd is a shell builtin              &lt;==看到了吗？ cd 是 shell 内置指令
```

通过 type 这个指令我们可以知道每个指令是否为 bash 的内置指令。 此外，由于利用 type 搜寻后面的名称时，如果后面接的名称并不能以可执行文件的状态被找到， 那么该名称是不会被显示出来的。也就是说， type 主要在找出“可执行文件”而不是一般文件文件名喔！ 呵呵！所以，这个 type 也可以用来作为类似 which 指令的用途啦！找指令用的！

10.1.6 指令的下达与快速编辑按钮

我们在第四章的开始下达指令小节已经提到过在 shell 环境下的指令下达方法，如果你忘记了请回到第四章再去回忆一下！这里不重复说明了。 鸟哥这里仅就反斜线 （\） 来说明一下指令下达的方式啰！

```
范例：如果指令串太长的话，如何使用两行来输出？
[dmtsai@study ~]$ cp /var/spool/mail/root /etc/crontab \
&gt; /etc/fstab /root
```

上面这个指令用途是将三个文件复制到 /root 这个目录下而已。不过，因为指令太长， 于是鸟哥就利用“ [Enter] ”来将 [Enter] 这个按键“跳脱！”开来，让 [Enter] 按键不再具有“开始执行”的功能！好让指令可以继续在下一行输入。 需要特别留意， [Enter] 按键是紧接着反斜线 （\） 的，两者中间没有其他字符。 因为 \ 仅跳脱“紧接着的下一个字符”而已！所以，万一我写成：“ \ [Enter] ”，亦即 [Enter] 与反斜线中间有一个空格时，则 \ 跳脱的是“空白键”而不是 [Enter] 按键！这个地方请再仔细的看一遍！很重要！

如果顺利跳脱 [Enter] 后，下一行最前面就会主动出现 > 的符号， 你可以继续输入指令啰！也就是说，那个 > 是系统自动出现的，你不需要输入。

另外，当你所需要下达的指令特别长，或者是你输入了一串错误的指令时，你想要快速的将这串指令整个删除掉，一般来说，我们都是按下删除键的。 有没有其他的快速组合键可以协助呢？是有的！常见的有下面这些：

组合键	功能与示范
[ctrl]+u/[ctrl]+k	分别是从光标处向前删除指令串 （[ctrl]+u） 及向后删除指令串 （[ctrl]+k） 。
[ctrl]+a/[ctrl]+e	分别是让光标移动到整个指令串的最前面 （[ctrl]+a） 或最后面 （[ctrl]+e） 。

总之，当我们顺利的在终端机 （tty） 上面登陆后， Linux 就会依据 /etc/passwd 文件的设置给我们一个 shell （默认是 bash），然后我们就可以依据上面的指令下达方式来操作 shell，之后，我们就可以通过 man 这个线上查询来查询指令的使用方式与参数说明， 很不错吧！那么我们就赶紧更进一步来操作 bash 这个好玩的东西啰！

10.2 Shell 的变量功能

变量是 bash 环境中非常重要的一个玩意儿，我们知道 Linux 是多用户多任务的环境，每个人登陆系统都能取得一个 bash shell， 每个人都能够使用 bash 下达 mail 这个指令来收受“自己”的邮件等等。问题是， bash 是如何得知你的邮件信箱是哪个文件？ 这就需要“变量”的帮助啦！所以，你说变量重不重要呢？下面我们将介绍重要的环境变量、变量的取用与设置等数据， 呼呼！动动脑时间又来到啰！^_^

10.2.1 什么是变量？

那么，什么是“变量”呢？简单的说，就是让某一个特定字串代表不固定的内容就是了。举个大家在国中都会学到的数学例子， 那就是：“ y = ax + b ”这东西，在等号左边的（y）就是变量，在等号右边的（ax+b）就是变量内容。 要注意的是，左边是未知数，右边是已知数喔！ 讲的更简单一点，我们可以“用一个简单的 "字眼" 来取代另一个比较复杂或者是容易变动的数据”。这有什么好处啊？最大的好处就是“方便！”。

- 变量的可变性与方便性

举例来说，我们每个帐号的邮件信箱默认是以 MAIL 这个变量来进行存取的， 当 dmtsai 这个使用者登陆时，他便会取得 MAIL 这个变量，而这个变量的内容其实就是 /var/spool/mail/dmtsai， 那如果 vbird 登陆呢？他取得的 MAIL 这个变量的内容其实就是 /var/spool/mail/vbird 。 而我们使用信件读取指令 mail 来读取自己的邮件信箱时，嘿嘿，这支程序可以直接读取 MAIL 这个变量的内容， 就能够自动的分辨出属于自己的信箱信件啰！这样一来，设计程序的设计师就真的很方便的啦！

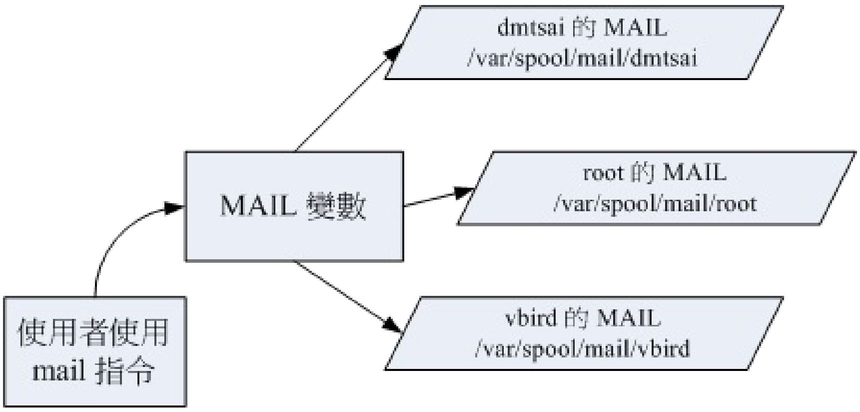


图10.2.1、程序、变量与不同使用者的关系

如上图所示，由于系统已经帮我们规划好 MAIL 这个变量，所以使用者只要知道 mail 这个指令如何使用即可， mail 会主动的取用 MAIL 这个变量，就能够如上图所示的取得自己的邮件信箱了！（注意大小写，小写的 mail 是指令， 大写的 MAIL 则是变量名称喔！）

那么使用变量真的比较好吗？这是当然的！想像一个例子，如果 mail 这个指令将 root 收信的邮件信箱 （mailbox） 文件名为 /var/spool/mail/root 直接写入程序码中。那么当 dmtsai 要使用 mail 时，将会取得 /var/spool/mail/root 这个文件的内容！ 不合理吧！所以你就需要帮 dmtsai 也设计一个 mail 的程序，将 /var/spool/mail/dmtsai 写死到 mail 的程序码当中！ 天呐！那系统要有多少个 mail 指令啊？反过来说，使用变量就变的很简单了！因为你不需要更动到程序码啊！ 只要将 MAIL 这个变量带入不同的内容即可让所有使用者通过 mail 取得自己的信件！当然简单多了！

- 影响 bash 环境操作的变量

某些特定变量会影响到 bash 的环境喔！举例来说，我们前面已经提到过很多次的那个 PATH 变量！ 你能不能在任何目录下执行某个指令，与 PATH 这个变量有很大的关系。例如你下达 ls 这个指令时，系统就是通过 PATH 这个变量里面的内容所记录的路径顺序来搜寻指令的呢！如果在搜寻完 PATH 变量内的路径还找不到 ls 这个指令时， 就会在屏幕上显示“ command not found ”的错误讯息了。

如果说的学理一点，那么由于在 Linux System 下面，所有的线程都是需要一个执行码， 而就如同上面提到的，你“真正以 shell 来跟 Linux 沟通，是在正确的登陆 Linux 之后！”这个时候你就有一个 bash 的执行程序，也才可以真正的经由 bash 来跟系统沟通啰！而在进入 shell 之前，也正如同上面提到的，由于系统需要一些变量来提供他数据的存取 （或者是一些环境的设置参数值， 例如是否要显示彩色等等的） ，所以就有一些所谓的“环境变量” 需要来读入系统中了！这些环境变量例如 PATH、HOME、MAIL、SHELL 等等，都是很重要的， 为了区别与自订变量的不同，环境变量通常以大写字符来表示呢！

- 脚本程序设计 （shell script） 的好帮手

这些还都只是系统默认的变量的目的，如果是个人的设置方面的应用呢：例如你要写一个大型的 script 时，有些数据因为可能由于使用者习惯的不同而有差异，比如说路径好了，由于该路径在 script 被使用在相当多的地方，如果下次换了一部主机，都要修改 script 里面的所有路径，那么我一定会疯掉！ 这个时候如果使用变量，而将该变量的定义写在最前面，后面相关的路径名称都以变量来取代， 嘿嘿！那么你只要修改一行就等于修改整篇 script 了！方便的很！所以，良好的程序设计师都会善用变量的定义！

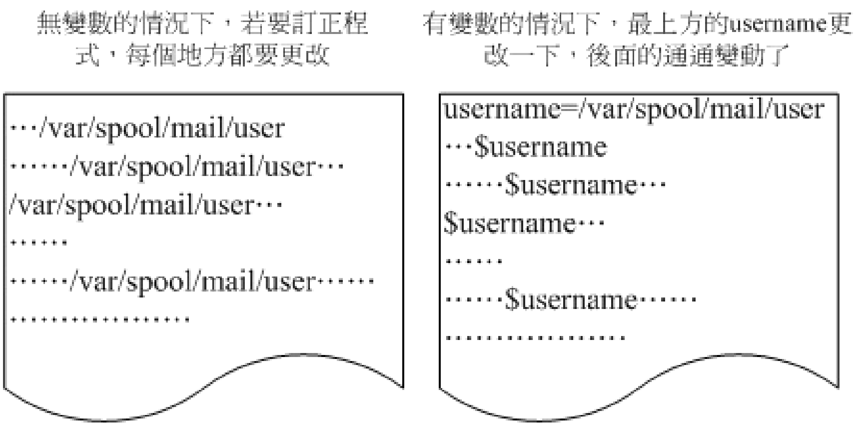


图10.2.2、变量应用于 shell script 的示意图

最后我们就简单的对“什么是变量”作个简单定义好了：“变量就是以一组文字或符号等，来取代一些设置或者是一串保留的数据！”， 例如：我设置了“myname”就是“VBird”，所以当你读取 myname 这个变量的时候，系统自然就会知道！哈！那就是 VBird 啦！ 那么如何“显示变量”呢？这就需要使用到 echo 这个指令啦！

10.2.2 变量的取用与设置：echo, 变量设置规则, unset

说的口沫横飞的，也不知道“变量”与“变量代表的内容”有啥关系？ 那我们就将“变量”的“内容”拿出来给您瞧瞧好了。你可以利用 echo 这个指令来取用变量， 但是，变量在被取用时，前面必须要加上钱字号“ $ ”才行，举例来说，要知道 PATH 的内容，该如何是好？

- 变量的取用: echo

```
[dmtsai@study ~]$ echo $variable
[dmtsai@study ~]$ echo $PATH
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/bi
[dmtsai@study ~]$ echo ${PATH}  # 近年来，鸟哥比较偏向使用这种格式喔！
```

变量的取用就如同上面的范例，利用 echo 就能够读出，只是需要在变量名称前面加上 $ ，或者是以 ${变量} 的方式来取用都可以！当然啦，那个 echo 的功能可是很多的， 我们这里单纯是拿 echo 来读出变量的内容而已，更多的 echo 使用，请自行给他 man echo 吧！ ^_^

例题：请在屏幕上面显示出您的环境变量 HOME 与 MAIL：答： `echo $HOME` 或者是 `echo ${HOME}` ， `echo $MAIL` 或者是 `echo ${MAIL}`

现在我们知道了变量与变量内容之间的相关性了，好了，那么我要如何“设置”或者是“修改” 某个变量的内容啊？很简单啦！用“等号（=）”连接变量与他的内容就好啦！举例来说： 我要将 myname 这个变量名称的内容设置为 VBird ，那么：

```
[dmtsai@study ~]$ echo ${myname}
       &lt;==这里并没有任何数据～因为这个变量尚未被设置！是空的！
[dmtsai@study ~]$ myname=VBird
[dmtsai@study ~]$ echo ${myname}
VBird  &lt;==出现了！因为这个变量已经被设置了！
```

瞧！如此一来，这个变量名称 myname 的内容就带有 VBird 这个数据啰～ 而由上面的例子当中，我们也可以知道： 在 bash 当中，当一个变量名称尚未被设置时，默认的内容是“空”的。 另外，变量在设置时，还是需要符合某些规定的，否则会设置失败喔！这些规则如下所示啊！



Tips 要请各位读者注意喔，每一种 shell 的语法都不相同～在变量的使用上，bash 在你没有设置的变量中强迫去 echo 时，它会显示出空的值。 在其他某些 shell 中，随便去 echo 一个不存在的变量，它是会出现错误讯息的喔！要注意！要注意！

- 变量的设置规则
- 变量与变量内容以一个等号“=”来链接，如下所示：“myname=VBird”
- 等号两边不能直接接空白字符，如下所示为错误：“myname = VBird”或“myname=VBird Tsai”
- 变量名称只能是英文字母与数字，但是开头字符不能是数字，如下为错误：“2myname=VBird”
- 变量内容若有空白字符可使用双引号“"”或单引号“'”将变量内容结合起来，但
    - 双引号内的特殊字符如 $ 等，可以保有原本的特性，如下所示：“var="lang is $LANG"”则“echo $var”可得“lang is zh_TW.UTF-8”
    - 单引号内的特殊字符则仅为一般字符 （纯文本），如下所示：“var='lang is $LANG'”则“echo $var”可得“lang is $LANG”
- 可用跳脱字符“ \ ”将特殊符号（如 [Enter], $, \, 空白字符, '等）变成一般字符，如：“myname=VBird\ Tsai”
- 在一串指令的执行中，还需要借由其他额外的指令所提供的信息时，可以使用反单引号“`指令`”或 “$（指令）”。特别注意，那个 ` 是键盘上方的数字键 1 左边那个按键，而不是单引号！ 例如想要取得核心版本的设置：“version=$（uname -r）”再“echo $version”可得“3.10.0-229.el7.x86_64”
- 若该变量为扩增变量内容时，则可用 "$变量名称" 或 ${变量} 累加内容，如下所示：“PATH="$PATH":/home/bin”或“PATH=${PATH}:/home/bin”
- 若该变量需要在其他子程序执行，则需要以 export 来使变量变成环境变量：“export PATH”
- 通常大写字符为系统默认变量，自行设置变量可以使用小写字符，方便判断 （纯粹依照使用者兴趣与嗜好） ；
- 取消变量的方法为使用 unset ：“unset 变量名称”例如取消 myname 的设置：“unset myname”

下面让鸟哥举几个例子来让你试看看，就知道怎么设置好你的变量啰！

```
范例一：设置一变量 name ，且内容为 VBird
[dmtsai@study ~]$ 12name=VBird
bash: 12name=VBird: command not found...   &lt;==屏幕会显示错误！因为不能以数字开头！
[dmtsai@study ~]$ name = VBird              &lt;==还是错误！因为有空白！
[dmtsai@study ~]$ name=VBird                &lt;==OK 的啦！

范例二：承上题，若变量内容为 VBird's name 呢，就是变量内容含有特殊符号时：
[dmtsai@study ~]$ name=VBird's name
# 单引号与双引号必须要成对，在上面的设置中仅有一个单引号，因此当你按下 enter 后，
# 你还可以继续输入变量内容。这与我们所需要的功能不同，失败啦！
# 记得，失败后要复原请按下 [ctrl]-c 结束！
[dmtsai@study ~]$ name="VBird's name"    &lt;==OK 的啦！
# 指令是由左边向右找→，先遇到的引号先有用，因此如上所示， 单引号变成一般字符！
[dmtsai@study ~]$ name='VBird's name'    &lt;==失败的啦！
# 因为前两个单引号已成对，后面就多了一个不成对的单引号了！因此也就失败了！
[dmtsai@study ~]$ name=VBird\'s\ name     &lt;==OK 的啦！
# 利用反斜线 （\） 跳脱特殊字符，例如单引号与空白键，这也是 OK 的啦！

范例三：我要在 PATH 这个变量当中“累加”:/home/dmtsai/bin 这个目录
[dmtsai@study ~]$ PATH=$PATH:/home/dmtsai/bin
[dmtsai@study ~]$ PATH="$PATH":/home/dmtsai/bin
[dmtsai@study ~]$ PATH=${PATH}:/home/dmtsai/bin
# 上面这三种格式在 PATH 里头的设置都是 OK 的！但是下面的例子就不见得啰！

范例四：承范例三，我要将 name 的内容多出 "yes" 呢？
[dmtsai@study ~]$ name=$nameyes
# 知道了吧？如果没有双引号，那么变量成了啥？name 的内容是 $nameyes 这个变量！
# 呵呵！我们可没有设置过 nameyes 这个变量呐！所以，应该是下面这样才对！
[dmtsai@study ~]$ name="$name"yes
[dmtsai@study ~]$ name=${name}yes  &lt;==以此例较佳！

范例五：如何让我刚刚设置的 name=VBird 可以用在下个 shell 的程序？
[dmtsai@study ~]$ name=VBird
[dmtsai@study ~]$ bash        &lt;==进入到所谓的子程序
[dmtsai@study ~]$ echo $name  &lt;==子程序：再次的 echo 一下；
       &lt;==嘿嘿！并没有刚刚设置的内容喔！
[dmtsai@study ~]$ exit        &lt;==子程序：离开这个子程序
[dmtsai@study ~]$ export name
[dmtsai@study ~]$ bash        &lt;==进入到所谓的子程序
[dmtsai@study ~]$ echo $name  &lt;==子程序：在此执行！
VBird  &lt;==看吧！出现设置值了！
[dmtsai@study ~]$ exit        &lt;==子程序：离开这个子程序
```

什么是“子程序”呢？就是说，在我目前这个 shell 的情况下，去启用另一个新的 shell ，新的那个 shell 就是子程序啦！在一般的状态下，父程序的自订变量是无法在子程序内使用的。但是通过 export 将变量变成环境变量后，就能够在子程序下面应用了！很不赖吧！至于程序的相关概念， 我们会在第十六章程序管理当中提到的喔！

```
范例六：如何进入到您目前核心的模块目录？
[dmtsai@study ~]$ cd /lib/modules/`uname -r`/kernel
[dmtsai@study ~]$ cd /lib/modules/$（uname -r）/kernel  # 以此例较佳！
```

每个 Linux 都能够拥有多个核心版本，且几乎 distribution 的核心版本都不相同。以 CentOS 7.1 （未更新前） 为例，他的默认核心版本是 3.10.0-229.el7.x86_64 ，所以核心模块目录在 /lib/modules/3.10.0-229.el7.x86_64/kernel/ 内。 也由于每个 distributions 的这个值都不相同，但是我们却可以利用 uname -r 这个指令先取得版本信息。所以啰，就可以通过上面指令当中的内含指令 $（uname -r） 先取得版本输出到 cd ... 那个指令当中，就能够顺利的进入目前核心的驱动程序所放置的目录啰！很方便吧！

其实上面的指令可以说是作了两次动作，亦即是：

1. 先进行反单引号内的动作“uname -r”并得到核心版本为 3.10.0-229.el7.x86_64
2. 将上述的结果带入原指令，故得指令为：“cd /lib/modules/3.10.0-229.el7.x86_64/kernel/”



Tips 为什么鸟哥比较建议记忆 $（ command ） 呢？还记得小时候学数学的加减乘除，我们都知道得要先乘除后加减。那如果硬要先加减再乘除呢？ 当然就是加上括号 （） 来处理即可啊！所以啰，这个指令的处理方式也差不多，只是括号左边得要加个钱字号就是了！

```
范例七：取消刚刚设置的 name 这个变量内容
[dmtsai@study ~]$ unset name
```

根据上面的案例你可以试试看！就可以了解变量的设置啰！这个是很重要的呦！请勤加练习！ 其中，较为重要的一些特殊符号的使用啰！例如单引号、双引号、跳脱字符、钱字号、反单引号等等，下面的例题想一想吧！

例题：在变量的设置当中，单引号与双引号的用途有何不同？ 答：单引号与双引号的最大不同在于双引号仍然可以保有变量的内容，但单引号内仅能是一般字符 ，而不会有特殊符号。我们以下面的例子做说明：假设您定义了一个变量， name=VBird ，现在想以 name 这个变量的内容定义出 myname 显示 VBird its me 这个内容，要如何订定呢？

> [dmtsai@study ~]$ name=VBird > [dmtsai@study ~]$ echo $name > VBird > [dmtsai@study ~]$ myname="$name its me" > [dmtsai@study ~]$ echo $myname > VBird its me > [dmtsai@study ~]$ myname='$name its me' > [dmtsai@study ~]$ echo $myname > $name its me

发现了吗？没错！使用了单引号的时候，那么 $name 将失去原有的变量内容，仅为一般字符的显示型态而已！这里必需要特别小心在意！

例题：在指令下达的过程中，反单引号（ `）这个符号代表的意义为何？答：在一串指令中，在` 之内的指令将会被先执行，而其执行出来的结果将做为外部的输入信息！例如 uname -r 会显示出目前的核心版本，而我们的核心版本在 /lib/modules 里面，因此，你可以先执行 uname -r 找出核心版本，然后再以“ cd 目录”到该目录下，当然也可以执行如同上面范例六的执行内容啰。

另外再举个例子，我们也知道， locate 指令可以列出所有的相关文件文件名，但是，如果我想要知道各个文件的权限呢？举例来说，我想要知道每个 crontab 相关文件名的权限：

> [dmtsai@study ~]$ ls -ld `locate crontab` > [dmtsai@study ~]$ ls -ld $（locate crontab）

如此一来，先以 locate 将文件名数据都列出来，再以 ls 指令来处理的意思啦！瞭了吗？ ^_^

例题：若你有一个常去的工作目录名称为："/cluster/server/work/taiwan_2015/003/"，如何进行该目录的简化？答：在一般的情况下，如果你想要进入上述的目录得要"cd /cluster/server/work/taiwan_2015/003/"， 以鸟哥自己的案例来说，鸟哥跑数值模式常常会设置很长的目录名称（避免忘记），但如此一来变换目录就很麻烦。 此时，鸟哥习惯利用下面的方式来降低指令下达错误的问题：

> [dmtsai@study ~]$ work="/cluster/server/work/taiwan_2015/003/" > [dmtsai@study ~]$ cd $work

未来我想要使用其他目录作为我的模式工作目录时，只要变更 work 这个变量即可！而这个变量又可以在 bash 的配置文件（~/.bashrc）中直接指定，那我每次登陆只要执行" cd $work "就能够去到数值模式仿真的工作目录了！是否很方便呢？ ^_^

10.2.3 环境变量的功能

环境变量可以帮我们达到很多功能～包括主文件夹的变换啊、提示字符的显示啊、可执行文件搜寻的路径啊等等的， 还有很多很多啦！那么，既然环境变量有那么多的功能，问一下，目前我的 shell 环境中， 有多少默认的环境变量啊？我们可以利用两个指令来查阅，分别是 env 与 export 呢！

- 用 env 观察环境变量与常见环境变量说明

```
范例一：列出目前的 shell 环境下的所有环境变量与其内容。
[dmtsai@study ~]$ env
HOSTNAME=study.centos.vbird    &lt;== 这部主机的主机名称
TERM=xterm                     &lt;== 这个终端机使用的环境是什么类型
SHELL=/bin/bash                &lt;== 目前这个环境下，使用的 Shell 是哪一个程序？
HISTSIZE=1000                  &lt;== "记录指令的笔数"在 CentOS 默认可记录 1000 笔
OLDPWD=/home/dmtsai            &lt;== 上一个工作目录的所在
LC_ALL=en_US.utf8              &lt;== 由于语系的关系，鸟哥偷偷丢上来的一个设置
USER=dmtsai                    &lt;== 使用者的名称啊！
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01
or=40;31;01:mi=01;05;37;41:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32
*.tar=01...                    &lt;== 一些颜色显示
MAIL=/var/spool/mail/dmtsai    &lt;== 这个使用者所取用的 mailbox 位置
PATH=/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmts
PWD=/home/dmtsai               &lt;== 目前使用者所在的工作目录 （利用 pwd 取出！）
LANG=zh_TW.UTF-8               &lt;== 这个与语系有关，下面会再介绍！
HOME=/home/dmtsai              &lt;== 这个使用者的主文件夹啊！
LOGNAME=dmtsai                 &lt;== 登陆者用来登陆的帐号名称
_=/usr/bin/env                 &lt;== 上一次使用的指令的最后一个参数（或指令本身）
```

env 是 environment （环境） 的简写啊，上面的例子当中，是列出来所有的环境变量。当然，如果使用 export 也会是一样的内容～ 只不过， export 还有其他额外的功能就是了，我们等一下再提这个 export 指令。 那么上面这些变量有些什么功用呢？下面我们就一个一个来分析分析！

- HOME 代表使用者的主文件夹。还记得我们可以使用 cd ~ 去到自己的主文件夹吗？或者利用 cd 就可以直接回到使用者主文件夹了。那就是取用这个变量啦～ 有很多程序都可能会取用到这个变量的值！

- SHELL 告知我们，目前这个环境使用的 SHELL 是哪支程序？ Linux 默认使用 /bin/bash 的啦！
- HISTSIZE 这个与“历史命令”有关，亦即是， 我们曾经下达过的指令可以被系统记录下来，而记录的“笔数”则是由这个值来设置的。
- MAIL 当我们使用 mail 这个指令在收信时，系统会去读取的邮件信箱文件 （mailbox）。
- PATH 就是可执行文件搜寻的路径啦～目录与目录中间以冒号（:）分隔， 由于文件的搜寻是依序由 PATH 的变量内的目录来查询，所以，目录的顺序也是重要的喔。
- LANG 这个重要！就是语系数据啰～很多讯息都会用到他， 举例来说，当我们在启动某些 perl 的程序语言文件时，他会主动的去分析语系数据文件， 如果发现有他无法解析的编码语系，可能会产生错误喔！一般来说，我们中文编码通常是 zh_TW.Big5 或者是 zh_TW.UTF-8，这两个编码偏偏不容易被解译出来，所以，有的时候，可能需要修订一下语系数据。 这部分我们会在下个小节做介绍的！
- RANDOM 这个玩意儿就是“随机乱数”的变量啦！目前大多数的 distributions 都会有乱数产生器，那就是 /dev/random 这个文件。 我们可以通过这个乱数文件相关的变量 （$RANDOM） 来随机取得乱数值喔。在 BASH 的环境下，这个 RANDOM 变量的内容，介于 0~32767 之间，所以，你只要 echo $RANDOM 时，系统就会主动的随机取出一个介于 0~32767 的数值。万一我想要使用 0~9 之间的数值呢？呵呵～利用 declare 宣告数值类型， 然后这样做就可以了：

```
[dmtsai@study ~]$ declare -i number=$RANDOM*10/32768 ; echo $number
8   &lt;== 此时会随机取出 0~9 之间的数值喔！
```

大致上是有这些环境变量啦～里面有些比较重要的参数，在下面我们都会另外进行一些说明的～

- 用 set 观察所有变量 （含环境变量与自订变量）

bash 可不只有环境变量喔，还有一些与 bash 操作接口有关的变量，以及使用者自己定义的变量存在的。 那么这些变量如何观察呢？这个时候就得要使用 set 这个指令了。 set 除了环境变量之外， 还会将其他在 bash 内的变量通通显示出来哩！信息很多，下面鸟哥仅列出几个重要的内容：

```
[dmtsai@study ~]$ set
BASH=/bin/bash                          &lt;== bash 的主程序放置路径
BASH_VERSINFO=([0]="4" [1]="2" [2]="46" [3]="1" [4]="release" [5]="x86_64-redhat-linux-gn
BASH_VERSION='4.2.46(1)-release'        &lt;== 这两行是 bash 的版本啊！
COLUMNS=90                              &lt;== 在目前的终端机环境下，使用的字段有几个字符长度
HISTFILE=/home/dmtsai/.bash_history     &lt;== 历史命令记录的放置文件，隐藏文件
HISTFILESIZE=1000                       &lt;== 存起来（与上个变量有关）的文件之指令的最大纪录笔数。
HISTSIZE=1000                           &lt;== 目前环境下，内存中记录的历史命令最大笔数。
IFS=$' \t\n'                            &lt;== 默认的分隔符号
LINES=20                                &lt;== 目前的终端机下的最大行数
MACHTYPE=x86_64-redhat-linux-gnu        &lt;== 安装的机器类型
OSTYPE=linux-gnu                        &lt;== 操作系统的类型！
PS1='[\u@\h \W]\$ '                     &lt;== PS1 就厉害了。这个是命令提示字符，也就是我们常见的
                                            [root@www ~]# 或 [dmtsai ~]$ 的设置值啦！可以更动的
PS2='&gt; '                                 &lt;== 如果你使用跳脱符号 （\） 第二行以后的提示字符也
$                                       &lt;== 目前这个 shell 所使用的 PID
?                                       &lt;== 刚刚执行完指令的回传值。
...
# 有许多可以使用的函数库功能被鸟哥取消啰！请自行查阅！
```

一般来说，不论是否为环境变量，只要跟我们目前这个 shell 的操作接口有关的变量， 通常都会被设置为大写字符，也就是说，“基本上，在 Linux 默认的情况中，使用{大写的字母}来设置的变量一般为系统内定需要的变量”。 OK！OK！那么上头那些变量当中，有哪些是比较重要的？大概有这几个吧！

- PS1：（提示字符的设置）

这是 PS1 （数字的 1 不是英文字母） ，这个东西就是我们的“命令提示字符”喔！ 当我们每次按下 [Enter] 按键去执行某个指令后，最后要再次出现提示字符时， 就会主动去读取这个变量值了。上头 PS1 内显示的是一些特殊符号，这些特殊符号可以显示不同的信息， 每个 distributions 的 bash 默认的 PS1 变量内容可能有些许的差异，不要紧，“习惯你自己的习惯”就好了。 你可以用 man bash [3]去查询一下 PS1 的相关说明，以理解下面的一些符号意义。

- \d ：可显示出“星期 月 日”的日期格式，如："Mon Feb 2"
- \H ：完整的主机名称。举例来说，鸟哥的练习机为“study.centos.vbird”
- \h ：仅取主机名称在第一个小数点之前的名字，如鸟哥主机则为“study”后面省略
- \t ：显示时间，为 24 小时格式的“HH:MM:SS”
- \T ：显示时间，为 12 小时格式的“HH:MM:SS”
- \A ：显示时间，为 24 小时格式的“HH:MM”
- \@ ：显示时间，为 12 小时格式的“am/pm”样式
- \u ：目前使用者的帐号名称，如“dmtsai”；
- \v ：BASH 的版本信息，如鸟哥的测试主机版本为 4.2.46（1）-release，仅取“4.2”显示
- \w ：完整的工作目录名称，由根目录写起的目录名称。但主文件夹会以 ~ 取代；
- \W ：利用 basename 函数取得工作目录名称，所以仅会列出最后一个目录名。
- # ：下达的第几个指令。
- $ ：提示字符，如果是 root 时，提示字符为 # ，否则就是 $ 啰～

好了，让我们来看看 CentOS 默认的 PS1 内容吧："[\u@\h \W]$ "，现在你知道那些反斜线后的数据意义了吧？ 要注意喔！那个反斜线后的数据为 PS1 的特殊功能，与 bash 的变量设置没关系啦！不要搞混了喔！ 那你现在知道为何你的命令提示字符是：" [dmtsai@study ~]$ "了吧？好了，那么假设我想要有类似下面的提示字符：

> [dmtsai@study /home/dmtsai 16:50 #12]$

那个 # 代表第 12 次下达的指令。那么应该如何设置 PS1 呢？可以这样啊：

```
[dmtsai@study ~]$ cd /home
[dmtsai@study home]$ PS1='[\u@\h \w \A #\#]\$ '
[dmtsai@study /home 17:02 #85]$
# 看到了吗？提示字符变了！变的很有趣吧！其中，那个 #85 比较有趣，
# 如果您再随便输入几次 ls 后，该数字就会增加喔！为啥？上面有说明滴！
```

- $：（关于本 shell 的 PID）

钱字号本身也是个变量喔！这个咚咚代表的是"目前这个 Shell 的线程代号"，亦即是所谓的 PID （Process ID）。 更多的程序观念，我们会在第四篇的时候提及。想要知道我们的 shell 的 PID ，就可以用：" echo $$ "即可！出现的数字就是你的 PID 号码。

- ?：（关于上个执行指令的回传值）

虾密？问号也是一个特殊的变量？没错！在 bash 里面这个变量可重要的很！ 这个变量是："上一个执行的指令所回传的值"， 上面这句话的重点是"上一个指令"与"回传值"两个地方。当我们执行某些指令时， 这些指令都会回传一个执行后的代码。一般来说，如果成功的执行该指令， 则会回传一个 0 值，如果执行过程发生错误，就会回传"错误代码"才对！一般就是以非为 0 的数值来取代。 我们以下面的例子来看看：

```
[dmtsai@study ~]$ echo $SHELL
/bin/bash                                  &lt;==可顺利显示！没有错误！
[dmtsai@study ~]$ echo $?
0                                          &lt;==因为没问题，所以回传值为 0
[dmtsai@study ~]$ 12name=VBird
bash: 12name=VBird: command not found...   &lt;==发生错误了！bash回报有问题
[dmtsai@study ~]$ echo $?
127                                        &lt;==因为有问题，回传错误代码（非为0）
# 错误代码回传值依据软件而有不同，我们可以利用这个代码来搜寻错误的原因喔！
[dmtsai@study ~]$ echo $?
0
# 咦！怎么又变成正确了？这是因为 "?" 只与"上一个执行指令"有关，
# 所以，我们上一个指令是执行" echo $? "，当然没有错误，所以是 0 没错！
```

- OSTYPE, HOSTTYPE, MACHTYPE：（主机硬件与核心的等级）

我们在第零章、计算机概论内的 CPU 等级说明中谈过 CPU ， 目前个人计算机的 CPU 主要分为 32/64 位，其中 32 位又可分为 i386, i586, i686，而 64 位则称为 x86_64。 由于不同等级的 CPU 指令集不太相同，因此你的软件可能会针对某些 CPU 进行最优化，以求取较佳的

软件性能。 所以软件就有 i386, i686 及 x86_64 之分。以目前 （2015） 的主流硬件来说，几乎都是 x86_64 的天下！ 因此 CentOS 7 开始，已经不支持 i386 相容模式的安装光盘了～哇呜！进步的太快了！

要留意的是，较高阶的硬件通常会向下相容旧有的软件，但较高阶的软件可能无法在旧机器上面安装！ 我们在第二章就曾说明过， 这里再强调一次，你可以在 x86_64 的硬件上安装 i386 的 Linux 操作系统，但是你无法在 i686 的硬件上安装 x86_64 的 Linux 操作系统！这点得要牢记在心！

- export： 自订变量转成环境变量

谈了 env 与 set 现在知道有所谓的环境变量与自订变量，那么这两者之间有啥差异呢？其实这两者的差异在于“ 该变量是否会被子程序所继续引用”啦！唔！那么啥是父程序？子程序？ 这就得要了解一下指令的下达行为了。

当你登陆 Linux 并取得一个 bash 之后，你的 bash 就是一个独立的程序，这个程序的识别使用的是一个称为程序识别码，被称为 PID 的就是。 接下来你在这个 bash 下面所下达的任何指令都是由这个 bash 所衍生出来的，那些被下达的指令就被称为子程序了。 我们可以用下面的图示来简单的说明一下父程序与子程序的概念：

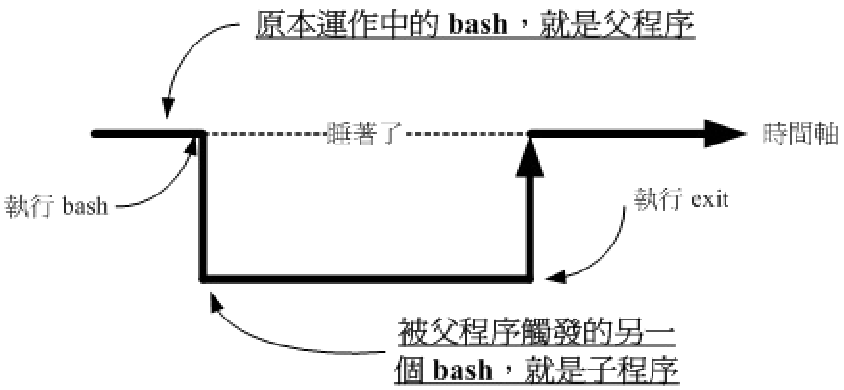


图10.2.3、程序相关性示意图

如上所示，我们在原本的 bash 下面执行另一个 bash ，结果操作的环境接口会跑到第二个 bash 去（就是子程序）， 那原本的 bash 就会在暂停的情况 （睡着了，就是 sleep） 。整个指令运行的环境是实线的部分！若要回到原本的 bash 去， 就只有将第二个 bash 结束掉 （下达 exit 或 logout） 才行。更多的程序概念我们会在第四篇谈及，这里只要有这个概念即可。

这个程序概念与变量有啥关系啊？关系可大了！因为子程序仅会继承父程序的环境变量， 子程序不会继承父程序的自订变量啦！所以你在原本 bash 的自订变量在进入了子程序后就会消失不见， 一直到你离开子程序并回到原本的父程序后，这个变量才会又出现！

换个角度来想，也就是说，如果我能将自订变量变成环境变量的话，那不就可以让该变量值继续存在于子程序了？ 呵呵！没错！此时，那个 export 指令就很有用啦！如你想要让该变量内容继续的在子程序中使用，那么就请执行：

```
[dmtsai@study ~]$ export 变量名称
```

这东西用在“分享自己的变量设置给后来调用的文件或其他程序”啦！ 像鸟哥常常在自己的主文件后面调用其他附属文件（类似函数的功能），但是主文件与附属文件内都有相同的变量名称， 若一再重复设置时，要修改也很麻烦，此时只要在原本的第一个文件内设置好“ export 变量 ”， 后面所调用的文件就能够使用这个变量设置了！而不需要重复设置，这非常实用于 shell script 当中喔！ 如果仅下达 export 而没有接变量时，那么此时将会把所有的“环境变量”秀出来喔！例如：

```
[dmtsai@study ~]$ export
declare -x HISTSIZE="1000"
declare -x HOME="/home/dmtsai"
declare -x HOSTNAME="study.centos.vbird"
declare -x LANG="zh_TW.UTF-8"
declare -x LC_ALL="en_US.utf8"
# 后面的鸟哥就都直接省略了！不然....浪费版面～ ^_^
```

那如何将环境变量转成自订变量呢？可以使用本章后续介绍的 declare 呢！

10.2.4 影响显示结果的语系变量 （locale）

还记得我们在第四章里面提到的语系问题吗？ 就是当我们使用 man command 的方式去查询某个数据的说明文档时，该说明文档的内容可能会因为我们使用的语系不同而产生乱码。 另外，利用 ls 查询文件的时间时，也可能会有乱码出现在时间的部分。那个问题其实就是语系的问题啦。

目前大多数的 Linux distributions 已经都是支持日渐流行的万国码了，也都支持大部分的国家语系。 那么我们的 Linux 到底支持了多少的语系呢？这可以由 locale 这个指令来查询到喔！

```
[dmtsai@study ~]$ locale -a
....（前面省略）....
zh_TW
zh_TW.big5     &lt;==大五码的中文编码
zh_TW.euctw
zh_TW.utf8     &lt;==万国码的中文编码
zu_ZA
zu_ZA.iso88591
zu_ZA.utf8
```

正体中文语系至少支持了两种以上的编码，一种是目前还是很常见的 big5 ，另一种则是越来越热门的 utf-8 编码。 那么我们如何修订这些编码呢？其实可以通过下面这些变量的说：

```
[dmtsai@study ~]$ locale  &lt;==后面不加任何选项与参数即可！
LANG=en_US                   &lt;==主语言的环境
LC_CTYPE="en_US"             &lt;==字符（文字）辨识的编码
LC_NUMERIC="en_US"           &lt;==数字系统的显示讯息
LC_TIME="en_US"              &lt;==时间系统的显示数据
LC_COLLATE="en_US"           &lt;==字串的比较与排序等
LC_MONETARY="en_US"          &lt;==币值格式的显示等
LC_MESSAGES="en_US"          &lt;==讯息显示的内容，如功能表、错误讯息等
LC_ALL=                      &lt;==整体语系的环境
....（后面省略）....
```

基本上，你可以逐一设置每个与语系有关的变量数据，但事实上，如果其他的语系变量都未设置， 且你有设置 LANG 或者是 LC_ALL 时，则其他的语系变量就会被这两个变量所取代！这也是为什么我们在 Linux 当中，通常说明仅设置 LANG 或 LC_ALL 这两个变量而已，因为他是最主要的设置变量！ 好了，那么你应该要觉得奇怪的是，为什么在 Linux 主机的终端机接口 （tty1 ~ tty6） 的环境下，如果设置“ LANG=zh_TW.utf8 ”这个设置值生效后，使用 man 或者其他讯息输出时， 都会有一堆乱码，尤其是使用 ls -l 这个参数时？

因为在 Linux 主机的终端机接口环境下是无法显示像中文这么复杂的编码文字， 所以就会产生乱码了。也就是如此，我们才会必须要在 tty1 ~ tty6 的环境下， 加装一些中文化接口的软件，才能够看到中文啊！不过，如果你是在 MS Windows 主机以远端连线服务器的软件连线到主机的话，那么，嘿嘿！其实命令行确实是可以看到中文的。 此时反而你得要在 LC_ALL 设置中文编码才好呢！



Tips 无论如何，如果发生一些乱码的问题，那么设置系统里面保有的语系编码， 例如： en_US 或 en_US.utf8 等等的设置，应该就 OK 的啦！好了，那么系统默认支持多少种语系呢？ 当我们使用 locale 时，系统是列出目前 Linux 主机内保有的语系文件， 这些语系文件都放置在： /usr/lib/locale/ 这个目录中。

你当然可以让每个使用者自己去调整自己喜好的语系，但是整体系统默认的语系定义在哪里呢？ 其实就是在 /etc/locale.conf 啰！这个文件在 CentOS 7.x 的内容有点像这样：

```
[dmtsai@study ~]$ cat /etc/locale.conf
LANG=zh_TW.utf8
LC_NUMERIC=zh_TW.UTF-8
LC_TIME=zh_TW.UTF-8
LC_MONETARY=zh_TW.UTF-8
LC_PAPER=zh_TW.UTF-8
LC_MEASUREMENT=zh_TW.UTF-8
```

因为鸟哥在第三章的安装时选择的是中文语系安装画面， 所以这个文件默认就会使用中文编码啦！你也可以自行将他改成你想要的语系编码即可。



Tips 假设你有一个纯文本原本是在 Windows 下面创建的，那么这个文件默认可能是 big5 的编码格式。 在你将这个文件上传到 Linux 主机后，在 X window 下面打开时，咦！怎么中文字通通变成乱码了？ 别担心！因为如上所示， Linux 目前大多默认是万国码显示嘛！你只要

将打开该文件的软件编码由 utf8 改成 big5 就能够看到正确的中文了！

例题：鸟哥原本是中文语系，所有显示的数据通通是中文。但为了网页显示的关系，需要将输出转成英文 （en_US.utf8） 的语系来展示才行。 但鸟哥又不想要写入配置文件！毕竟是暂时显示用的～那该如何处理？答：其实不很难，重点是 LANG 及 LC_ALL 而已！但在 CentOS 7 当中，你要让 LC_ALL 生效时，得要使用 export 转成环境变量才行耶！ 所以就是这样搞：

```
[dmtsai@study ~]$ locale
LANG=zh_TW.UTF-8
LC_CTYPE="zh_TW.UTF-8"
LC_NUMERIC="zh_TW.UTF-8"
LC_TIME="zh_TW.UTF-8"

[dmtsai@study ~]$ LANG=en_US.utf8; locale
[dmtsai@study ~]$ export LC_ALL=en_US.utf8; locale  # 你就会看到与上头有不同的语系啰！
```

10.2.5 变量的有效范围

虾密？变量也有使用的“范围”？没错啊～我们在上头的 export 指令说明中，就提到了这个概念了。如果在跑程序的时候，有父程序与子程序的不同程序关系时， 则“变量”可否被引用与 export 有关。被 export 后的变量，我们可以称他为“环境变量”！ 环境变量可以被子程序所引用，但是其他的自订变量内容就不会存在于子程序中。



Tips 在某些不同的书籍会谈到“全域变量, global variable”与“区域变量, local variable”。 在鸟哥的这个章节中，基本上你可以这样看待： 环境变量=全域变量 自订变量=区域变量

在学理方面，为什么环境变量的数据可以被子程序所引用呢？这是因为内存配置的关系！理论上是这样的：

- 当启动一个 shell，操作系统会分配一记忆区块给 shell 使用，此内存内之变量可让子程序取用
- 若在父程序利用 export 功能，可以让自订变量的内容写到上述的记忆区块当中（环境变量）；
- 当载入另一个 shell 时 （亦即启动子程序，而离开原本的父程序了），子 shell 可以将父 shell 的环境变量所在的记忆区块导入自己的环境变量区块当中。

通过这样的关系，我们就可以让某些变量在相关的程序之间存在，以帮助自己更方便的操作环境喔！ 不过要提醒的是，这个“环境变量”与“bash 的操作环境”意思不太一样，举例来说， PS1 并不是环境变量， 但是这个 PS1 会影响到 bash 的接口 （提示字符嘛）！相关性要厘清喔！^_^

10.2.6 变量键盘读取、阵列与宣告： read, array, declare

我们上面提到的变量设置功能，都是由命令行直接设置的，那么，可不可以让使用者能够经由键盘输入？ 什么意思呢？是否记得某些程序执行的过程当中，会等待使用者输入 "yes/no" 之类的讯息啊？ 在 bash 里面也有相对应的功能喔！此外，我们还可以宣告这个变量的属性，例如：阵列或者是数字等等的。下面就来看看吧！

- read

要读取来自键盘输入的变量，就是用 read 这个指令了。这个指令最常被用在 shell script 的撰写当中， 想要跟使用者对谈？用这个指令就对了。关于 script 的写法，我们会在第十三章介绍，下面先来瞧一瞧 read 的相关语法吧！

```
[dmtsai@study ~]$ read [-pt] variable
选项与参数：
-p  ：后面可以接提示字符！
-t  ：后面可以接等待的“秒数！”这个比较有趣～不会一直等待使用者啦！

范例一：让使用者由键盘输入一内容，将该内容变成名为 atest 的变量
[dmtsai@study ~]$ read atest
This is a test        &lt;==此时光标会等待你输入！请输入左侧文字看看
[dmtsai@study ~]$ echo ${atest}
This is a test          &lt;==你刚刚输入的数据已经变成一个变量内容！

范例二：提示使用者 30 秒内输入自己的大名，将该输入字串作为名为 named 的变量内容
[dmtsai@study ~]$ read -p "Please keyin your name: " -t 30 named
Please keyin your name: VBird Tsai   &lt;==注意看，会有提示字符喔！
[dmtsai@study ~]$ echo ${named}
VBird Tsai        &lt;==输入的数据又变成一个变量的内容了！
```

read 之后不加任何参数，直接加上变量名称，那么下面就会主动出现一个空白行等待你的输入（如范例一）。 如果加上 -t 后面接秒数，例如上面的范例二，那么 30 秒之内没有任何动作时， 该指令就会自动略过了～如果是加上 -p ，嘿嘿！在输入的光标前就会有比较多可以用的提示字符给我们参考！ 在指令的下达里面，比较美观啦！ ^_^

- declare / typeset

declare 或 typeset 是一样的功能，就是在“宣告变量的类型”。如果使用 declare 后面并没有接任何参数，那么 bash 就会主动的将所有的变量名称与内容通通叫出来，就好像使用 set 一样啦！ 那么 declare 还有什么语法呢？看看先：

```
[dmtsai@study ~]$ declare [-aixr] variable
选项与参数：
-a  ：将后面名为 variable 的变量定义成为阵列 （array） 类型
-i  ：将后面名为 variable 的变量定义成为整数数字 （integer） 类型
-x  ：用法与 export 一样，就是将后面的 variable 变成环境变量；
-r  ：将变量设置成为 readonly 类型，该变量不可被更改内容，也不能 unset

范例一：让变量 sum 进行 100+300+50 的加总结果
[dmtsai@study ~]$ sum=100+300+50
[dmtsai@study ~]$ echo ${sum}
100+300+50  &lt;==咦！怎么没有帮我计算加总？因为这是文字体态的变量属性啊！
[dmtsai@study ~]$ declare -i sum=100+300+50
[dmtsai@study ~]$ echo ${sum}
450         &lt;==瞭乎？？
```

由于在默认的情况下面， bash 对于变量有几个基本的定义：

- 变量类型默认为“字串”，所以若不指定变量类型，则 1+2 为一个“字串”而不是“计算式”。所以上述第一个执行的结果才会出现那个情况的；
- bash 环境中的数值运算，默认最多仅能到达整数形态，所以 1/3 结果是 0；

现在你晓得为啥你需要进行变量宣告了吧？如果需要非字串类型的变量，那就得要进行变量的宣告才行啦！ 下面继续来玩些其他的 declare 功能。

```
范例二：将 sum 变成环境变量
[dmtsai@study ~]$ declare -x sum
[dmtsai@study ~]$ export &#124; grep sum
declare -ix sum="450"  &lt;==果然出现了！包括有 i 与 x 的宣告！

范例三：让 sum 变成只读属性，不可更动！
[dmtsai@study ~]$ declare -r sum
[dmtsai@study ~]$ sum=tesgting
-bash: sum: readonly variable  &lt;==老天爷～不能改这个变量了！

范例四：让 sum 变成非环境变量的自订变量吧！
[dmtsai@study ~]$ declare +x sum  &lt;== 将 - 变成 + 可以进行“取消”动作
[dmtsai@study ~]$ declare -p sum  &lt;== -p 可以单独列出变量的类型
declare -ir sum="450" &lt;== 看吧！只剩下 i, r 的类型，不具有 x 啰！
```

declare 也是个很有用的功能～尤其是当我们需要使用到下面的阵列功能时， 他也可以帮我们宣告阵列的属性喔！不过，老话一句，阵列也是在 shell script 比较常用的啦！ 比较有趣的是，如果你不小心将变量设置为“只读”，通常得要登出再登陆才能复原该变量的类型了！ @_@

- 阵列 （array） 变量类型

某些时候，我们必须使用阵列来宣告一些变量，这有什么好处啊？在一般人的使用上， 果然是看不出来有什么好处的！不过，如果您曾经写过程序的话，那才会比较了解阵列的意义～ 阵列对写数值程序的设计师来说，可是不能错过学习的重点之一哩！好！不啰唆～ 那么要如何设置阵列的变量与内容呢？在 bash 里头，阵列的设置方式是：

> var[index]=content

意思是说，我有一个阵列名称为 var ，而这个阵列的内容为 var[1]=小明， var[2]=大明，var[3]=好明 .... 等等，那个 index 就是一些数字啦，重点是用中刮号 （[ ]） 来设置的。 目前我们 bash 提供的是一维阵列。老实说，如果您不必写一些复杂的程序， 那么这个阵列的地方，可以先略过，等到有需要再来学习即可！因为要制作出阵列， 通常与循环或者其他判断式交互使用才有比较高的存在意义！

```
范例：设置上面提到的 var[1] ～ var[3] 的变量。
[dmtsai@study ~]$ var[1]="small min"
[dmtsai@study ~]$ var[2]="big min"
[dmtsai@study ~]$ var[3]="nice min"
[dmtsai@study ~]$ echo "${var[1]}, ${var[2]}, ${var[3]}"
small min, big min, nice min
```

阵列的变量类型比较有趣的地方在于“读取”，一般来说，建议直接以 ${阵列} 的方式来读取，比较正确无误的啦！这也是为啥鸟哥一开始就建议你使用 ${变量} 来记忆的原因！

10.2.7 与文件系统及程序的限制关系： ulimit

想像一个状况：我的 Linux 主机里面同时登陆了十个人，这十个人不知怎么搞的， 同时打开了 100 个文件，每个文件的大小约 10MBytes ，请问一下， 我的 Linux 主机的内存要有多大才够？ 10*100*10 = 10000 MBytes = 10GBytes ... 老天爷，这样，系统不挂点才有鬼哩！为了要预防这个情况的发生，所以我们的 bash 是可以“限制使用者的某些系统资源”的，包括可以打开的文件数量， 可以使用的 CPU 时间，可以使用的内存总量等等。如何设置？用 ulimit 吧！

```
[dmtsai@study ~]$ ulimit [-SHacdfltu] [配额]
选项与参数：
-H  ：hard limit ，严格的设置，必定不能超过这个设置的数值；
-S  ：soft limit ，警告的设置，可以超过这个设置值，但是若超过则有警告讯息。
      在设置上，通常 soft 会比 hard 小，举例来说，soft 可设置为 80 而 hard
      设置为 100，那么你可以使用到 90 （因为没有超过 100），但介于 80~100 之间时，
      系统会有警告讯息通知你！
-a  ：后面不接任何选项与参数，可列出所有的限制额度；
-c  ：当某些程序发生错误时，系统可能会将该程序在内存中的信息写成文件（除错用），
      这种文件就被称为核心文件（core file）。此为限制每个核心文件的最大容量。
-f  ：此 shell 可以创建的最大文件大小（一般可能设置为 2GB）单位为 KBytes
-d  ：程序可使用的最大断裂内存（segment）容量；
-l  ：可用于锁定 （lock） 的内存量
-t  ：可使用的最大 CPU 时间 （单位为秒）
-u  ：单一使用者可以使用的最大程序（process）数量。

范例一：列出你目前身份（假设为一般帐号）的所有限制数据数值
[dmtsai@study ~]$ ulimit -a
core file size          （blocks, -c） 0          &lt;==只要是 0 就代表没限制
data seg size           （kBytes, -d） unlimited
scheduling priority             （-e） 0
file size               （blocks, -f） unlimited  &lt;==可创建的单一文件的大小
pending signals                 （-i） 4903
max locked memory       （kBytes, -l） 64
max memory size         （kBytes, -m） unlimited
open files                      （-n） 1024       &lt;==同时可打开的文件数量
pipe size            （512 Bytes, -p） 8
POSIX message queues     （Bytes, -q） 819200
real-time priority              （-r） 0
stack size              （kBytes, -s） 8192
cpu time               （seconds, -t） unlimited
max user processes              （-u） 4096
virtual memory          （kBytes, -v） unlimited
file locks                      （-x） unlimited

范例二：限制使用者仅能创建 10MBytes 以下的容量的文件
[dmtsai@study ~]$ ulimit -f 10240
[dmtsai@study ~]$ ulimit -a &#124; grep 'file size'
core file size          （blocks, -c） 0
file size               （blocks, -f） 10240 &lt;==最大量为10240Kbyes，相当10MBytes

[dmtsai@study ~]$ dd if=/dev/zero of=123 bs=1M count=20
File size limit exceeded （core dumped） &lt;==尝试创建 20MB 的文件，结果失败了！

[dmtsai@study ~]$ rm 123  &lt;==赶快将这个文件删除啰！同时你得要登出再次的登陆才能解开 10M 的限制
```

还记得我们在第七章 Linux 磁盘文件系统里面提到过，单一 filesystem 能够支持的单一文件大小与 block 的大小有关。但是文件系统的限制容量都允许的太大了！如果想要让使用者创建的文件不要太大时， 我们是可以考虑用 ulimit 来限制使用者可以创建的文件大小喔！利用 ulimit -f 就可以来设置了！例如上面的范例二，要注意单位喔！单位是 KBytes。 若改天你一直无法创建一个大容量的文件，记得瞧一瞧 ulimit 的信息喔！



Tips 想要复原 ulimit 的设置最简单的方法就是登出再登陆，否则就是得要重新以 ulimit 设置才行！ 不过，要注意的是，一般身份使用者如果以 ulimit 设置了 -f 的文件大小， 那么他“只能继续减小文件大小，不能增加文件大小喔！”另外，若想要管控使用者的 ulimit 限值， 可以参考第十三章的 pam 的介绍。

10.2.8 变量内容的删除、取代与替换（Optional）

变量除了可以直接设置来修改原本的内容之外，有没有办法通过简单的动作来将变量的内容进行微调呢？ 举例来说，进行变量内容的删除、取代与替换等！是可以的！我们可以通过几个简单的小步骤来进行变量内容的微调喔！ 下面就来试试看！

- 变量内容的删除与取代

变量的内容可以很简单的通过几个咚咚来进行删除喔！我们使用 PATH 这个变量的内容来做测试好了。 请你依序进行下面的几个例子来玩玩，比较容易感受的到鸟哥在这里想要表达的意义：

```
范例一：先让小写的 path 自订变量设置的与 PATH 内容相同
[dmtsai@study ~]$ path=${PATH}
[dmtsai@study ~]$ echo ${path}
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/bi

范例二：假设我不喜欢 local/bin，所以要将前 1 个目录删除掉，如何显示？
[dmtsai@study ~]$ echo ${path#/*local/bin:}
/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/bin
```

上面这个范例很有趣的！他的重点可以用下面这张表格来说明：

```
${variable#/*local/bin:}
  上面的特殊字体部分是关键字！用在这种删除模式所必须存在的

${variable#/*local/bin:}
  这就是原本的变量名称，以上面范例二来说，这里就填写 path 这个“变量名称”啦！

${variable#/*local/bin:}
  这是重点！代表“从变量内容的最前面开始向右删除”，且仅删除最短的那个

${variable#/*local/bin:}
  代表要被删除的部分，由于 # 代表由前面开始删除，所以这里便由开始的 / 写起。
    需要注意的是，我们还可以通过万用字符 * 来取代 0 到无穷多个任意字符

  以上面范例二的结果来看， path 这个变量被删除的内容如下所示：
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/bi
```

很有趣吧！这样了解了 # 的功能了吗？接下来让我们来看看下面的范例三！

```
范例三：我想要删除前面所有的目录，仅保留最后一个目录
[dmtsai@study ~]$ echo ${path#/*:}
/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/bin
# 由于一个 # 仅删除掉最短的那个，因此他删除的情况可以用下面的删除线来看：
# /usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/

[dmtsai@study ~]$ echo ${path##/*:}
/home/dmtsai/bin
# 嘿！多加了一个 # 变成 ## 之后，他变成“删除掉最长的那个数据”！亦即是：
# /usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/
```

非常有趣！不是吗？因为在 PATH 这个变量的内容中，每个目录都是以冒号“:”隔开的， 所以要从头删除掉目录就是介于斜线 （/） 到冒号 （:） 之间的数据！但是 PATH 中不止一个冒号 （:） 啊！ 所以 # 与 ## 就分别代表：

- ：符合取代文字的“最短的”那一个；

- ：符合取代文字的“最长的”那一个

上面谈到的是“从前面开始删除变量内容”，那么如果想要“从后面向前删除变量内容”呢？ 这个时候就得使用百分比 （%） 符号了！来看看范例四怎么做吧！

```
范例四：我想要删除最后面那个目录，亦即从 : 到 bin 为止的字串
[dmtsai@study ~]$ echo ${path%:*bin}
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin
# 注意啊！最后面一个目录不见去！
# 这个 % 符号代表由最后面开始向前删除！所以上面得到的结果其实是来自如下：
# /usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/

范例五：那如果我只想要保留第一个目录呢？
[dmtsai@study ~]$ echo ${path%%:*bin}
/usr/local/bin
# 同样的， %% 代表的则是最长的符合字串，所以结果其实是来自如下：
# /usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/
```

由于我是想要由变量内容的后面向前面删除，而我这个变量内容最后面的结尾是“/home/dmtsai/bin”， 所以你可以看到上面我删除的数据最终一定是“bin”，亦即是“*:bin”*那个代表万用字符！ 至于 % 与 %% 的意义其实与 # 及 ## 类似！这样理解否？

例题：假设你是 dmtsai ，那你的 MAIL 变量应该是 /var/spool/mail/dmtsai 。假设你只想要保留最后面那个文件名 （dmtsai）， 前面的目录名称都不要了，如何利用 $MAIL 变量来达成？
答：题意其实是这样“/var/spool/mail/dmtsai”，亦即删除掉两条斜线间的所有数据（最长符合）。 这个时候你就可以这样做即可：

```
[dmtsai@study ~]$ echo ${MAIL##/*/}
```

相反的，如果你只想要拿掉文件名，保留目录的名称，亦即是“/var/spool/mail/dmtsai” （最短符合）。但假设你并不知道结尾的字母为何，此时你可以利用万用字符来处理即可，如下所示：

```
[dmtsai@study ~]$ echo ${MAIL%/*}
```

了解了删除功能后，接下来谈谈取代吧！继续玩玩范例六啰！

```
范例六：将 path 的变量内容内的 sbin 取代成大写 SBIN：
[dmtsai@study ~]$ echo ${path/sbin/SBIN}
/usr/local/bin:/usr/bin:/usr/local/SBIN:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/bi
# 这个部分就容易理解的多了！关键字在于那两个斜线，两斜线中间的是旧字串
# 后面的是新字串，所以结果就会出现如上述的特殊字体部分啰！

[dmtsai@study ~]$ echo ${path//sbin/SBIN}
/usr/local/bin:/usr/bin:/usr/local/SBIN:/usr/SBIN:/home/dmtsai/.local/bin:/home/dmtsai/bi
# 如果是两条斜线，那么就变成所有符合的内容都会被取代喔！
```

我们将这部份作个总结说明一下：

变量设置方式	说明
`${变量#关键字}` `${变量##关键字}`	若变量内容从头开始的数据符合“关键字”，则将符合的最短数据删除 若变量内容从头开始的数据符合“关键字”，则将符合的最长数据删除
`${变量%关键字}` `${变量%%关键字}`	若变量内容从尾向前的数据符合“关键字”，则将符合的最短数据删除 若变量内容从尾向前的数据符合“关键字”，则将符合的最长数据删除
`${变量/旧字串/新字串}` `${变量//旧字串/新字串}`	若变量内容符合“旧字串”则“第一个旧字串会被新字串取代” 若变量内容符合“旧字串”则“全部的旧字串会被新字串取代”

- 变量的测试与内容替换

在某些时刻我们常常需要“判断”某个变量是否存在，若变量存在则使用既有的设置，若变量不存在则给予一个常用的设置。 我们举下面的例子来说明好了，看看能不能较容易被你所理解呢！

```
范例一：测试一下是否存在 username 这个变量，若不存在则给予 username 内容为 root
[dmtsai@study ~]$ echo ${username}
           &lt;==由于出现空白，所以 username 可能不存在，也可能是空字串
[dmtsai@study ~]$ username=${username-root}
[dmtsai@study ~]$ echo ${username}
root       &lt;==因为 username 没有设置，所以主动给予名为 root 的内容。
[dmtsai@study ~]$ username="vbird tsai" &lt;==主动设置 username 的内容
[dmtsai@study ~]$ username=${username-root}
[dmtsai@study ~]$ echo ${username}
vbird tsai &lt;==因为 username 已经设置了，所以使用旧有的设置而不以 root 取代
```

在上面的范例中，重点在于减号“ - ”后面接的关键字！基本上你可以这样理解：

```
new_var=${old_var-content}
   新的变量，主要用来取代旧变量。新旧变量名称其实常常是一样的

new_var=${old_var-content}
   这是本范例中的关键字部分！必须要存在的哩！

new_var=${old_var-content}
   旧的变量，被测试的项目！

new_var=${old_var-content}
   变量的“内容”，在本范例中，这个部分是在“给予未设置变量的内容”
```

不过这还是有点问题！因为 username 可能已经被设置为“空字串”了！果真如此的话，那你还可以使用下面的范例来给予 username 的内容成为 root 喔！

```
范例二：若 username 未设置或为空字串，则将 username 内容设置为 root
[dmtsai@study ~]$ username=""
[dmtsai@study ~]$ username=${username-root}
[dmtsai@study ~]$ echo ${username}
      &lt;==因为 username 被设置为空字串了！所以当然还是保留为空字串！
[dmtsai@study ~]$ username=${username:-root}
[dmtsai@study ~]$ echo ${username}
root  &lt;==加上“ : ”后若变量内容为空或者是未设置，都能够以后面的内容替换！
```

在大括号内有没有冒号“:”的差别是很大的！加上冒号后，被测试的变量未被设置或者是已被设置为空字串时， 都能够用后面的内容 （本例中是使用 root 为内容） 来替换与设置！这样可以了解了吗？除了这样的测试之外， 还有其他的测试方法喔！鸟哥将他整理如下：



Tips 下面的例子当中，那个 var 与 str 为变量，我们想要针对 str 是否有设置来决定 var 的值喔！ 一般来说， str: 代表“str 没设置或为空的字串时”；至于 str 则仅为“没有该变量”。

变量设置方式	**str** 没有设置	**str** 为空字串	**str** 已设置非为空字串
var=${str-expr}	var=expr	var=	var=$str
var=${str:-expr}	var=expr	var=expr	var=$str
var=${str+expr}	var=	var=expr	var=expr
var=${str:+expr}	var=	var=	var=expr
var=${str=expr}	str=expr var=expr	str 不变 var=	str 不变 var=$str
var=${str:=expr}	str=expr var=expr	str=expr var=expr	str 不变 var=$str
var=${str?expr}	expr 输出至 stderr	var=	var=$str
var=${str:?expr}	expr 输出至 stderr	expr 输出至 stderr	var=$str

根据上面这张表，我们来进行几个范例的练习吧！ ^_^！首先让我们来测试一下，如果旧变量（str） 不存在时， 我们要给予新变量一个内容，若旧变量存在则新变量内容以旧变量来替换，结果如下：

```
测试：先假设 str 不存在 （用 unset） ，然后测试一下减号 （-） 的用法：
[dmtsai@study ~]$ unset str; var=${str-newvar}
[dmtsai@study ~]$ echo "var=${var}, str=${str}"
var=newvar, str=        &lt;==因为 str 不存在，所以 var 为 newvar

测试：若 str 已存在，测试一下 var 会变怎样？：
[dmtsai@study ~]$ str="oldvar"; var=${str-newvar}
[dmtsai@study ~]$ echo "var=${var}, str=${str}"
var=oldvar, str=oldvar  &lt;==因为 str 存在，所以 var 等于 str 的内容
```

关于减号 （-） 其实上面我们谈过了！这里的测试只是要让你更加了解，这个减号的测试并不会影响到旧变量的内容。 如果你想要将旧变量内容也一起替换掉的话，那么就使用等号（=） 吧！

```
测试：先假设 str 不存在 （用 unset） ，然后测试一下等号 （=） 的用法：
[dmtsai@study ~]$ unset str; var=${str=newvar}
[dmtsai@study ~]$ echo "var=${var}, str=${str}"
var=newvar, str=newvar  &lt;==因为 str 不存在，所以 var/str 均为 newvar

测试：如果 str 已存在了，测试一下 var 会变怎样？
[dmtsai@study ~]$ str="oldvar"; var=${str=newvar}
[dmtsai@study ~]$ echo "var=${var}, str=${str}"
var=oldvar, str=oldvar  &lt;==因为 str 存在，所以 var 等于 str 的内容
```

那如果我只是想知道，如果旧变量不存在时，整个测试就告知我“有错误”，此时就能够使用问号“？”的帮忙啦！ 下面这个测试练习一下先！

```
测试：若 str 不存在时，则 var 的测试结果直接显示 "无此变量"
[dmtsai@study ~]$ unset str; var=${str?无此变量}
-bash: str: 无此变量    &lt;==因为 str 不存在，所以输出错误讯息

测试：若 str 存在时，则 var 的内容会与 str 相同！
[dmtsai@study ~]$ str="oldvar"; var=${str?novar}
[dmtsai@study ~]$ echo "var=${var}, str=${str}"
var=oldvar, str=oldvar  &lt;==因为 str 存在，所以 var 等于 str 的内容
```

基本上这种变量的测试也能够通过 shell script 内的 if...then... 来处理， 不过既然 bash 有提供这么简单的方法来测试变量，那我们也可以多学一些嘛！ 不过这种变量测试通常是在程序设计当中比较容易出现，如果这里看不懂就先略过，未来有用到判断变量值时，再回来看看吧！ ^_^

10.3 命令别名与历史命令

我们知道在早期的 DOS 年代，清除屏幕上的信息可以使用 cls 来清除，但是在 Linux 里面，我们则是使用 clear 来清除画面的。那么可否让 cls 等于 clear 呢？可以啊！用啥方法？ link file 还是什么的？别急！下面我们介绍不用 link file 的命令别名来达成。那么什么又是历史命令？ 曾经做过的举动我们可以将他记录下来喔！那就是历史命令啰～下面分别来谈一谈这两个玩意儿。

10.3.1 命令别名设置： alias, unalias

命令别名是一个很有趣的东西，特别是你的惯用指令特别长的时候！还有， 增设默认的选项在一些惯用的指令上面，可以预防一些不小心误杀文件的情况发生的时候！ 举个例子来说，如果你要查询隐藏文件，并且需要长的列出与一页一页翻看，那么需要下达“ ls -al | more ”这个指令，鸟哥是觉得很烦啦！ 要输入好几个单字！那可不可以使用 lm 来简化呢？当然可以，你可以在命令列下面下达：

```
[dmtsai@study ~]$ alias lm='ls -al &#124; more'
```

立刻多出了一个可以执行的指令喔！这个指令名称为 lm ，且其实他是执行 ls -al | more 啊！真是方便。不过， 要注意的是：“alias 的定义规则与变量定义规则几乎相同”， 所以你只要在 alias 后面加上你的 {“别名”='指令 选项...' }， 以后你只要输入 lm 就相当于输入了 ls -al|more 这一串指令！很方便吧！

另外，命令别名的设置还可以取代既有的指令喔！举例来说，我们知道 root 可以移除 （rm） 任何数据！所以当你以 root 的身份在进行工作时，需要特别小心， 但是总有失手的时候，那么 rm 提供了一个选项来让我们确认是否要移除该文件，那就是 -i 这个选项！所以，你可以这样做：

```
[dmtsai@study ~]$ alias rm='rm -i'
```

那么以后使用 rm 的时候，就不用太担心会有错误删除的情况了！这也是命令别名的优点啰！那么如何知道目前有哪些的命令别名呢？就使用 alias 呀！

```
[dmtsai@study ~]$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias lm='ls -al &#124; more'
alias ls='ls --color=auto'
alias rm='rm -i'
alias vi='vim'
alias which='alias &#124; /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
```

由上面的数据当中，你也会发现一件事情啊，我们在第九章的 vim 程序编辑器里面提到 vi 与 vim 是不太一样的，vim 可以多作一些额外的语法检验与颜色显示。一般用户会有 vi=vim 的命令别名，但是 root 则是单纯使用 vi 而已。 如果你想要使用 vi 就直接以 vim 来打开文件的话，使用" alias vi='vim' "这个设置即可。 至于如果要取消命令别名的话，那么就使用 unalias 吧！例如要将刚刚的 lm 命令别名拿掉，就使用：

```
[dmtsai@study ~]$ unalias lm
```

那么命令别名与变量有什么不同呢？命令别名是"新创一个新的指令， 你可以直接下达该指令"的，至于变量则需要使用类似" echo "指令才能够调用出变量的内容！ 这两者当然不一样！很多初学者在这里老是搞不清楚！要注意啊！ ^_^

例题：DOS 年代，列出目录与文件就是 dir ，而清除屏幕就是 cls ，那么如果我想要在 linux 里面也使用相同的指令呢？答：很简单，通过 clear 与 ls 来进行命令别名的创建：

> alias cls='clear' > alias dir='ls -l'

10.3.2 历史命令：history

前面我们提过 bash 有提供指令历史的服务！那么如何查询我们曾经下达过的指令呢？就使用 history 啰！当然，如果觉得 histsory 要输入的字符太多太麻烦，可以使用命令别名来设置呢！ 不要跟我说还不会设置呦！ ^_^

```
[dmtsai@study ~]$ alias h='history'
```

如此则输入 h 等于输入 history 啰！好了，我们来谈一谈 history 的用法吧！

```
[dmtsai@study ~]$ history [n]
[dmtsai@study ~]$ history [-c]
[dmtsai@study ~]$ history [-raw] histfiles
选项与参数：
n   ：数字，意思是“要列出最近的 n 笔命令列表”的意思！
-c  ：将目前的 shell 中的所有 history 内容全部消除
-a  ：将目前新增的 history 指令新增入 histfiles 中，若没有加 histfiles ，
      则默认写入 ~/.bash_history
-r  ：将 histfiles 的内容读到目前这个 shell 的 history 记忆中；
-w  ：将目前的 history 记忆内容写入 histfiles 中！

范例一：列出目前内存内的所有 history 记忆
[dmtsai@study ~]$ history
# 前面省略
 1017  man bash
 1018  ll
 1019  history
 1020  history
# 列出的信息当中，共分两栏，第一栏为该指令在这个 shell 当中的代码，
# 另一个则是指令本身的内容喔！至于会秀出几笔指令记录，则与 HISTSIZE 有关！

范例二：列出目前最近的 3 笔数据
[dmtsai@study ~]$ history 3
 1019  history
 1020  history
 1021  history 3

范例三：立刻将目前的数据写入 histfile 当中
[dmtsai@study ~]$ history -w
# 在默认的情况下，会将历史纪录写入 ~/.bash_history 当中！
[dmtsai@study ~]$ echo ${HISTSIZE}
1000
```

在正常的情况下，历史命令的读取与记录是这样的：

- 当我们以 bash 登陆 Linux 主机之后，系统会主动的由主文件夹的 ~/.bash_history 读取以前曾经下过的指令，那么 ~/.bash_history 会记录几笔数据呢？这就与你 bash 的 HISTFILESIZE 这个变量设置值有关了！
- 假设我这次登陆主机后，共下达过 100 次指令，“等我登出时， 系统就会将 101~1100 这总共 1000 笔历史命令更新到 ~/.bash_history 当中。” 也就是说，历史命令在我登出时，会将最近的 HISTFILESIZE 笔记录到我的纪录档当中啦！
- 当然，也可以用 history -w 强制立刻写入的！那为何用“更新”两个字呢？ 因为 ~/.bash_history 记录的笔数永远都是 HISTFILESIZE 那么多，旧的讯息会被主动的拿掉！ 仅保留最新的！

那么 history 这个历史命令只可以让我查询命令而已吗？呵呵！当然不止啊！ 我们可以利用相关的功能来帮我们执行命令呢！举例来说啰：

```
[dmtsai@study ~]$ !number
[dmtsai@study ~]$ !command
[dmtsai@study ~]$ !!
选项与参数：
number   ：执行第几笔指令的意思；
command  ：由最近的指令向前搜寻“指令串开头为 command”的那个指令，并执行；
!!       ：就是执行上一个指令（相当于按↑按键后，按 Enter）

[dmtsai@study ~]$ history
   66  man rm
   67  alias
   68  man history
   69  history
[dmtsai@study ~]$ !66  &lt;==执行第 66 笔指令
[dmtsai@study ~]$ !!   &lt;==执行上一个指令，本例中亦即 !66
[dmtsai@study ~]$ !al  &lt;==执行最近以 al 为开头的指令（上头列出的第 67 个）
```

经过上面的介绍，瞭乎？历史命令用法可多了！如果我想要执行上一个指令， 除了使用上下键之外，我可以直接以“ !! ” 来下达上个指令的内容，此外， 我也可以直接选择下达第 n 个指令，“ !n ”来执行，也可以使用指令标头，例如 “ !vi ”来执行最近指令开头是 vi 的命令行！相当的方便而好用！

基本上 history 的用途很大的！但是需要小心安全的问题！尤其是 root 的历史纪录文件，这是 Cracker 的最爱！因为不小心的 root 会将很多的重要数据在执行的过程中会被纪录在 ~/.bash_history 当中，如果这个文件被解析的话，后果不堪吶！无论如何，使用 history 配合“ ! ”曾经使用过的指令下达是很有效率的一个指令下达方法！

- 同一帐号同时多次登陆的 history 写入问题

有些朋友在练习 linux 的时候喜欢同时开好几个 bash 接口，这些 bash 的身份都是 root 。 这样会有 ~/.bash_history 的写入问题吗？想一想，因为这些 bash 在同时以 root 的身份登陆， 因此所有的 bash 都有自己的 1000 笔记录在内存中。因为等到登出时才会更新记录文件，所以啰， 最后登出的那个 bash 才会是最后写入的数据。唔！如此一来其他 bash 的指令操作就不会被记录下来了 （其实有被记录，只是被后来的最后一个 bash 所覆盖更新了） 。

由于多重登陆有这样的问题，所以很多朋友都习惯单一 bash 登陆，再用工作控制 （job control, 第四篇会介绍） 来切换不同工作！ 这样才能够将所有曾经下达过的指令记录下来，也才方便未来系统管理员进行指令的 debug 啊！

- 无法记录时间

历史命令还有一个问题，那就是无法记录指令下达的时间。由于这 1000 笔历史命令是依序记录的， 但是并没有记录时间，所以在查询方面会有一些不方便。如果读者们有兴趣，其实可以通过 ~/.bash_logout 来进行 history 的记录，并加上 date 来增加时间参数，也是一个可以应用的方向喔！ 有兴趣的朋友可以先看看情境仿真题一吧！



Tips 鸟哥经常需要设计线上题目给学生考试用，所以需要登陆系统去设计环境，设计完毕后再将该硬盘分派给学生来考试使用。 只是，经常很担心同学不小心输入 history 就会得知鸟哥要考试的重点文件与指令，因此就得要使用 history -c; history -w 来强迫更新纪录档了！ 提供给您参考！

10.4 Bash Shell 的操作环境：

是否记得我们登陆主机的时候，屏幕上头会有一些说明文字，告知我们的 Linux 版本啊什么的， 还有，登陆的时候我们还可以给予使用者一些讯息或者欢迎文字呢。此外， 我们习惯的环境变量、命令别名等等的，是否可以登陆就主动的帮我设置好？ 这些都是需要注意的。另外，这些设置值又可以分为系统整体设置值与各人喜好设置值， 仅是一些文件放置的地点不同啦！这我们后面也会来谈一谈的！

10.4.1 路径与指令搜寻顺序

我们在第五章与第六章都曾谈过“相对路径与绝对路径”的关系， 在本章的前几小节也谈到了 alias 与 bash 的内置命令。现在我们知道系统里面其实有不少的 ls 指令， 或者是包括内置的 echo 指令，那么来想一想，如果一个指令 （例如 ls） 被下达时， 到底是哪一个 ls 被拿来运行？很有趣吧！基本上，指令运行的顺序可以这样看：

1. 以相对/绝对路径执行指令，例如“ /bin/ls ”或“ ./ls ”；
2. 由 alias 找到该指令来执行；
3. 由 bash 内置的 （builtin） 指令来执行；
4. 通过 $PATH 这个变量的顺序搜寻到的第一个指令来执行。

举例来说，你可以下达 /bin/ls 及单纯的 ls 看看，会发现使用 ls 有颜色但是 /bin/ls 则没有颜色。 因为 /bin/ls 是直接取用该指令来下达，而 ls 会因为“ alias ls='ls --color=auto' ”这个命令别名而先使用！ 如果想要了解指令搜寻的顺序，其实通过 type -a ls 也可以查询的到啦！上述的顺序最好先了解喔！

例题：设置 echo 的命令别名成为 echo -n ，然后再观察 echo 执行的顺序答：

```
[dmtsai@study ~]$ alias echo='echo -n'
[dmtsai@study ~]$ type -a echo
echo is aliased to `echo -n'
echo is a shell builtin
echo is /usr/bin/echo
```

瞧！很清楚吧！先 alias 再 builtin 再由 $PATH 找到 /bin/echo 啰！

10.4.2 bash 的进站与欢迎讯息： /etc/issue, /etc/motd

虾密！ bash 也有进站画面与欢迎讯息喔？真假？真的啊！ 还记得在终端机接口 （tty1 ~ tty6） 登陆的时候，会有几行提示的字串吗？那就是进站画面啊！ 那个字串写在哪里啊？呵呵！在 /etc/issue 里面啊！先来看看：

```
[dmtsai@study ~]$ cat /etc/issue
\S
Kernel \r on an \m
```

鸟哥是以完全未更新过的 CentOS 7.1 作为范例，里面默认有三行，较有趣的地方在于 \r 与 \m。 就如同 $PS1 这变量一样，issue 这个文件的内容也是可以使用反斜线作为变量取用喔！你可以 man issue 配合 man agetty 得到下面的结果：

issue 内的各代码意义
\d 本地端时间的日期；
\l 显示第几个终端机接口；
\m 显示硬件的等级（i386/i486/i586/i686...）；
\n 显示主机的网络名称；
\O 显示 domain name；
\r 操作系统的版本（相当于 uname -r）
\t 显示本地端时间的时间；
\S 操作系统的名称；
\v 操作系统的版本。

做一下下面这个练习，看看能不能取得你要的进站画面？

例题：如果你在 tty3 的进站画面看到如下显示，该如何设置才能得到如下画面？

> CentOS Linux 7（Core）（terminal: tty3） > Date: 2015-07-08 17:29:19 > Kernel 3.10.0-229.el7.x86_64 on an x86_64 > Welcome!

注意，tty3 在不同的 tty 有不同显示，日期则是再按下 [enter] 后就会所有不同。答：很简单，用 root 的身份，并参考上述的反斜线功能去修改 /etc/issue 成为如下模样即可（共五行）：

```
\S （terminal: \l）
Date: \d \t
Kernel \r on an \m
Welcome!
```

曾有鸟哥的学生在这个 /etc/issue 内修改数据，光是利用简单的英文字母作出属于他自己的进站画面， 画面里面有他的中文名字呢！非常厉害！也有学生做成类似很大一个“囧”在进站画面，都非常有趣！

你要注意的是，除了 /etc/issue 之外还有个 /etc/issue.net 呢！这是啥？这个是提供给 telnet 这个远端登陆程序用的。 当我们使用 telnet 连接到主机时，主机的登陆画面就会显示 /etc/issue.net 而不是 /etc/issue 呢！

至于如果您想要让使用者登陆后取得一些讯息，例如您想要让大家都知道的讯息， 那么可以将讯息加入 /etc/motd 里面去！例如：当登陆后，告诉登陆者， 系统将会在某个固定时间进行维护工作，可以这样做 （一定要用 root 的身份才能修改喔！）：

```
[root@study ~]# vim /etc/motd
Hello everyone,
Our server will be maintained at 2015/07/10 0:00 ~ 24:00.
Please don't login server at that time. ^_^
```

那么当你的使用者（包括所有的一般帐号与 root）登陆主机后，就会显示这样的讯息出来：

```
Last login: Wed Jul  8 23:22:25 2015 from 127.0.0.1
Hello everyone,
Our server will be maintained at 2015/07/10 0:00 ~ 24:00.
Please don't login server at that time. ^_^
```

10.4.3 bash 的环境配置文件

你是否会觉得奇怪，怎么我们什么动作都没有进行，但是一进入 bash 就取得一堆有用的变量了？ 这是因为系统有一些环境设置文件的存在，让 bash 在启动时直接读取这些配置文件，以规划好 bash 的操作环境啦！ 而这些配置文件又可以分为全体系统的配置文件以及使用者个人偏好配置文件。要注意的是， 我们前几个小节谈到的命令别名啦、自订的变量啦，在你登出 bash 后就会失效，所以你想要保留你的设置， 就得要将这些设置写入配置文件才行。下面就让我们来聊聊吧！

- login 与 non-login shell

在开始介绍 bash 的配置文件前，我们一定要先知道的就是 login shell 与 non-login shell！ 重点在于有没有登陆 （login） 啦！

- login shell：取得 bash 时需要完整的登陆流程的，就称为 login shell。举例来说，你要由 tty1 ~ tty6 登陆，需要输入使用者的帐号与密码，此时取得的 bash 就称为“ login shell ”啰；
- non-login shell：取得 bash 接口的方法不需要重复登陆的举动，举例来说，（1）你以 X window 登陆 Linux 后， 再以 X 的图形化接口启动终端机，此时那个终端接口并没有需要再次的输入帐号与密码，那个 bash 的环境就称为 non-login shell了。（2）你在原本的 bash 环境下再次下达 bash 这个指令，同样的也没有输入帐号密码， 那第二个 bash （子程序） 也是 non-login shell 。

为什么要介绍 login, non-login shell 呢？这是因为这两个取得 bash 的情况中，读取的配置文件数据并不一样所致。 由于我们需要登陆系统，所以先谈谈 login shell 会读取哪些配置文件？一般来说，login shell 其实只会读取这两个配置文件：

1. /etc/profile：这是系统整体的设置，你最好不要修改这个文件；
2. ~/.bash_profile 或 ~/.bash_login 或 ~/.profile：属于使用者个人设置，你要改自己的数

据，就写入这里！

那么，就让我们来聊一聊这两个文件吧！这两个文件的内容可是非常繁复的喔！

- /etc/profile （login shell 才会读）

你可以使用 vim 去阅读一下这个文件的内容。这个配置文件可以利用使用者的识别码（UID） 来决定很多重要的变量数据， 这也是每个使用者登陆取得 bash 时一定会读取的配置文件！ 所以如果你想要帮所有使用者设置整体环境，那就是改这里啰！不过，没事还是不要随便改这个文件喔 这个文件设置的变量主要有：

- PATH：会依据 UID 决定 PATH 变量要不要含有 sbin 的系统指令目录；
- MAIL：依据帐号设置好使用者的 mailbox 到 /var/spool/mail/帐号名；
- USER：根据使用者的帐号设置此一变量内容；
- HOSTNAME：依据主机的 hostname 指令决定此一变量内容；
- HISTSIZE：历史命令记录笔数。CentOS 7.x 设置为 1000 ；
- umask：包括 root 默认为 022 而一般用户为 002 等！

/etc/profile 可不止会做这些事而已，他还会去调用外部的设置数据喔！在 CentOS 7.x 默认的情况下，下面这些数据会依序的被调用进来：

- /etc/profile.d/*.sh

其实这是个目录内的众多文件！只要在 /etc/profile.d/ 这个目录内且扩展名为 .sh ，另外，使用者能够具有 r 的权限， 那么该文件就会被 /etc/profile 调用进来。在 CentOS 7.x 中，这个目录下面的文件规范了 bash 操作接口的颜色、 语系、ll 与 ls 指令的命令别名、vi 的命令别名、which 的命令别名等等。如果你需要帮所有使用者设置一些共享的命令别名时， 可以在这个目录下面自行创建扩展名为 .sh 的文件，并将所需要的数据写入即可喔！

- /etc/locale.conf

这个文件是由 /etc/profile.d/lang.sh 调用进来的！这也是我们决定 bash 默认使用何种语系的重要配置文件！ 文件里最重要的就是 LANG/LC_ALL 这些个变量的设置啦！我们在前面的 locale 讨论过这个文件啰！ 自行回去瞧瞧先！

- /usr/share/bash-completion/completions/*

记得我们上头谈过 [tab] 的妙用吧？除了命令补齐、文件名补齐之外，还可以进行指令的选项/参数补齐功能！那就是从这个目录里面找到相对应的指令来处理的！ 其实这个目录下面的内容是由 /etc/profile.d/bash_completion.sh 这个文件载入的啦！

反正你只要记得，bash 的 login shell 情况下所读取的整体环境配置文件其实只有 /etc/profile，但是 /etc/profile 还会调用出其他的配置文件，所以让我们的 bash 操作接口变的非常的友善啦！ 接下来，让我们来瞧瞧，那么个人偏好的配置文件又是怎么回事？

- ~/.bash_profile （login shell 才会读）

bash 在读完了整体环境设置的 /etc/profile 并借此调用其他配置文件后，接下来则是会读取使用者的个人配置文件。 在 login shell 的 bash 环境中，所读取的个人偏好配置文件其实主要有三个，依序分别是：

1. ~/.bash_profile
2. ~/.bash_login
3. ~/.profile

其实 bash 的 login shell 设置只会读取上面三个文件的其中一个， 而读取的顺序则是依照上面的顺序。也就是说，如果 ~/.bash_profile 存在，那么其他两个文件不论有无存在，都不会被读取。 如果 ~/.bash_profile 不存在才会去读取 ~/.bash_login，而前两者都不存在才会读取 ~/.profile 的意思。 会有这么多的文件，其实是因应其他 shell 转换过来的使用者的习惯而已。 先让我们来看一下 dmtsai 的 /home/dmtsai/.bash_profile 的内容是怎样呢？

```
[dmtsai@study ~]$ cat ~/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then    &lt;==下面这三行在判断并读取 ~/.bashrc
        . ~/.bashrc
fi

# User specific environment and startup programs
PATH=$PATH:$HOME/.local/bin:$HOME/bin    &lt;==下面这几行在处理个人化设置
export PATH
```

这个文件内有设置 PATH 这个变量喔！而且还使用了 export 将 PATH 变成环境变量呢！ 由于 PATH 在 /etc/profile 当中已经设置过，所以在这里就以累加的方式增加使用者主文件夹下的 ~/bin/ 为额外的可执行文件放置目录。这也就是说，你可以将自己创建的可执行文件放置到你自己主文件夹下的 ~/bin/ 目录啦！ 那就可以直接执行该可执行文件而不需要使用绝对/相对路径来执行该文件。

这个文件的内容比较有趣的地方在于 if ... then ... 那一段！那一段程序码我们会在第十二章 shell script 谈到，假设你现在是看不懂的。 该段的内容指的是“判断主文件夹下的 ~/.bashrc 存在否，若存在则读入 ~/.bashrc 的设置”。 bash 配置文件的读入方式比较有趣，主要是通过一个指令“ source ”来读取的！ 也就是说 ~/.bash_profile 其实会再调用 ~/.bashrc 的设置内容喔！最后，我们来看看整个 login shell 的读取流程：

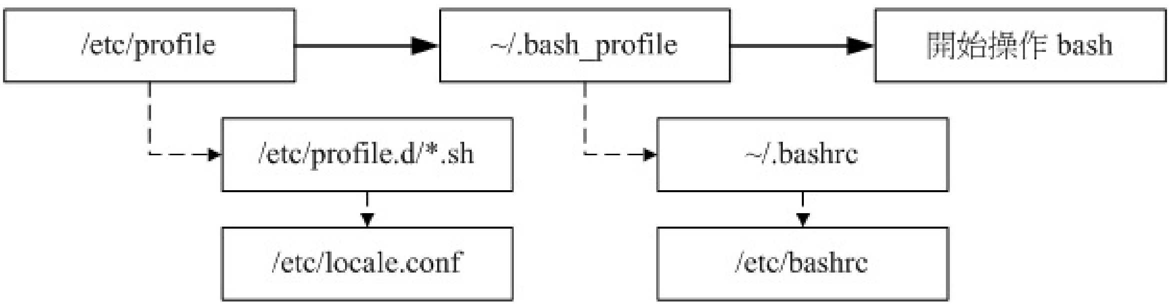


图 10.4.1、login shell 的配置文件读取流程

实线的的方向是主线流程，虚线的方向则是被调用的配置文件！从上面我们也可以清楚的知道，在 CentOS 的 login shell 环境下，最终被读取的配置文件是“ ~/.bashrc ”这个文件喔！所以，你当然可以将自己的偏好设置写入该文件即可。 下面我们还要讨论一下 source 与 ~/.bashrc 喔！

- source ：读入环境配置文件的指令

由于 /etc/profile 与 ~/.bash_profile 都是在取得 login shell 的时候才会读取的配置文件，所以， 如果你将自己的偏好设置写入上述的文件后，通常都是得登出再登陆后，该设置才会生效。那么，能不能直接读取配置文件而不登出登陆呢？ 可以的！那就得要利用 source 这个指令了！

```
[dmtsai@study ~]$ source 配置文件文件名

范例：将主文件夹的 ~/.bashrc 的设置读入目前的 bash 环境中
[dmtsai@study ~]$ source ~/.bashrc  &lt;==下面这两个指令是一样的！
[dmtsai@study ~]$  .  ~/.bashrc
```

利用 source 或小数点 （.） 都可以将配置文件的内容读进来目前的 shell 环境中！ 举例来说，我修改了 ~/.bashrc ，那么不需要登出，立即以 source ~/.bashrc 就可以将刚刚最新设置的内容读进来目前的环境中！很不错吧！还有，包括 ~/bash_profile 以及 /etc/profile 的设置中， 很多时候也都是利用到这个 source （或小数点） 的功能喔！

有没有可能会使用到不同环境配置文件的时候？有啊！ 最常发生在一个人的工作环境分为多种情况的时候了！举个例子来说，在鸟哥的大型主机中， 常常需要负责两到三个不同的案子，每个案子所需要处理的环境变量订定并不相同， 那么鸟哥就将这两三个案子分别编写属于该案子的环境变量设置文件，当需要该环境时，就直接“ source 变量文件 ”，如此一来，环境变量的设置就变的更简便而灵活了！

- ~/.bashrc （non-login shell 会读）

谈完了 login shell 后，那么 non-login shell 这种非登陆情况取得 bash 操作接口的环境配置文件又是什么？ 当你取得 non-login shell 时，该 bash 配置文件仅会读取 ~/.bashrc 而已啦！那么默认的 ~/.bashrc 内容是如何？

```
[root@study ~]# cat ~/.bashrc
# .bashrc

# User specific aliases and functions
alias rm='rm -i'             &lt;==使用者的个人设置
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then  &lt;==整体的环境设置
        . /etc/bashrc
fi
```

特别注意一下，由于 root 的身份与一般使用者不同，鸟哥是以 root 的身份取得上述的数据，如果是一般使用者的 ~/.bashrc 会有些许不同。看一下，你会发现在 root 的 ~/.bashrc 中其实已经规范了较为保险的命令别名了。 此外，咱们的 CentOS 7.x 还会主动的调用 /etc/bashrc 这个文件喔！为什么需要调用 /etc/bashrc 呢？ 因为 /etc/bashrc 帮我们的 bash 定义出下面的数据：

- 依据不同的 UID 规范出 umask 的值；
- 依据不同的 UID 规范出提示字符 （就是 PS1 变量）；
- 调用 /etc/profile.d/*.sh 的设置

你要注意的是，这个 /etc/bashrc 是 CentOS 特有的 （其实是 Red Hat 系统特有的），其他不同的 distributions 可能会放置在不同的文件名就是了。由于这个 ~/.bashrc 会调用 /etc/bashrc 及 /etc/profile.d/*.sh ， 所以，万一你没有 ~/.bashrc （可能自己不小心将他删除了），那么你会发现你的 bash 提示字符可能会变成这个样子：

```
-bash-4.2$
```

不要太担心啦！这是正常的，因为你并没有调用 /etc/bashrc 来规范 PS1 变量啦！而且这样的情况也不会影响你的 bash 使用。 如果你想要将命令提示字符捉回来，那么可以复制 /etc/skel/.bashrc 到你的主文件夹，再修订一下你所想要的内容， 并使用 source 去调用 ~/.bashrc ，那你的命令提示字符就会回来啦！

- 其他相关配置文件

事实上还有一些配置文件可能会影响到你的 bash 操作的，下面就来谈一谈：

- /etc/man_db.conf

这个文件乍看之下好像跟 bash 没相关性，但是对于系统管理员来说， 却也是很重要的一个文件！这的文件的内容“规范了使用 man 的时候， man page 的路径到哪里去寻找！”所以说的简单一点，这个文件规定了下达 man 的时候，该去哪里查看数据的路径设置！

那么什么时候要来修改这个文件呢？如果你是以 tarball 的方式来安装你的数据，那么你的 man page 可能会放置在 /usr/local/softpackage/man 里头，那个 softpackage 是你的套件名称， 这个时候你就得以手动的方式将该路径加到 /etc/man_db.conf 里头，否则使用 man 的时候就会找不到相关的说明文档啰。

- ~/.bash_history

还记得我们在历史命令提到过这个文件吧？默认的情况下， 我们的历史命令就记录在这里啊！而这个文件能够记录几笔数据，则与 HISTFILESIZE 这个变量有关啊。每次登陆 bash 后，bash 会先读取这个文件，将所有的历史指令读入内存， 因此，当我们登陆 bash 后就可以查知上次使用过哪些指令啰。至于更多的历史指令， 请自行回去参考喔！

- ~/.bash_logout

这个文件则记录了“当我登出 bash 后，系统再帮我做完什么动作后才离开”的意思。 你可以去读取一下这个文件的内容，默认的情况下，登出时， bash 只是帮我们清掉屏幕的讯息而已。 不过，你也可以将一些备份或者是其他你认为重要的工作写在这个文件中 （例如清空暂存盘）， 那么当你离开 Linux 的时候，就可以解决一些烦人的事情啰！

10.4.4 终端机的环境设置： stty, set

我们在第四章首次登陆 Linux 时就提过，可以在 tty1 ~ tty6 这六个命令行的终端机（terminal） 环境中登陆，登陆的时候我们可以取得一些字符设置的功能喔！ 举例来说，我们可以利用倒退键 （backspace，就是那个←符号的按键） 来删除命令列上的字符， 也可以使用 [ctrl]+c 来强制终止一个指令的运行，当输入错误时，就会有声音跑出来警告。这是怎么办到的呢？ 很简单啊！因为登陆终端机的时候，会自动的取得一些终端机的输入环境的设置啊！

事实上，目前我们使用的 Linux distributions 都帮我们作了最棒的使用者环境了， 所以大家可以不用担心操作环境的问题。不过，在某些 Unix like 的机器中，还是可能需要动用一些手脚， 才能够让我们的输入比较快乐～举例来说，利用 [backspace] 删除，要比利用 [Del] 按键来的顺手吧！ 但是某些 Unix 偏偏是以 [del] 来进行字符的删除啊！所以，这个时候就可以动动手脚啰～

那么如何查阅目前的一些按键内容呢？可以利用 stty （setting tty 终端机的意思） 呢！ stty 也可以帮助设置终端机的输入按键代表意义喔！

```
[dmtsai@study ~]$ stty [-a]
选项与参数：
-a  ：将目前所有的 stty 参数列出来；

范例一：列出所有的按键与按键内容
[dmtsai@study ~]$ stty -a
speed 38400 baud; rows 20; columns 90; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = &lt;undef&gt;; eol2 = &lt;un
swtch = &lt;undef&gt;; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext =
flush = ^O; min = 1; time = 0;
....（以下省略）....
```

我们可以利用 stty -a 来列出目前环境中所有的按键列表，在上头的列表当中，需要注意的是特殊字体那几个， 此外，如果出现 ^ 表示 [Ctrl] 那个按键的意思。举例来说， intr = ^C 表示利用 [ctrl] + c 来达成的。几个重要的代表意义是：

- intr : 送出一个 interrupt （中断） 的讯号给目前正在 run 的程序 （就是终止啰！）；
- quit : 送出一个 quit 的讯号给目前正在 run 的程序；
- erase : 向后删除字符，
- kill : 删除在目前命令行上的所有文字；
- eof : End of file 的意思，代表“结束输入”。
- start : 在某个程序停止后，重新启动他的 output
- stop : 停止目前屏幕的输出；

- susp : 送出一个 terminal stop 的讯号给正在 run 的程序。

记不记得我们在第四章讲过几个 Linux 热键啊？没错！ 就是这个 stty 设置值内的 intr（[ctrl]+c） / eof（[ctrl]+d） 啰～至于删除字符，就是 erase 那个设置值啦！ 如果你想要用 [ctrl]+h 来进行字符的删除，那么可以下达：

```
[dmtsai@study ~]$ stty erase ^h   # 这个设置看看就好，不必真的实做！不然还要改回来！
```

那么从此之后，你的删除字符就得要使用 [ctrl]+h 啰，按下 [backspace] 则会出现 ^? 字样呢！ 如果想要回复利用 [backspace] ，就下达 stty erase ^? 即可啊！ 至于更多的 stty 说明，记得参考一下 man stty 的内容喔！

问：因为鸟哥的工作经常在 Windows/Linux 之间切换，在 windows 下面，很多软件默认的储存快捷按钮是 [crtl]+s ，所以鸟哥习惯按这个按钮来处理。 不过，在 Linux 下面使用 vim 时，却也经常不小心就按下 [crtl]+s ！问题来了，按下这个组合钮之后，整个 vim 就不能动了 （整个画面锁死）！ 请问鸟哥该如何处置？答：参考一下 stty -a 的输出中，有个 stop 的项目就是按下 [crtl]+s 的！那么恢复成 start 就是 [crtl]+q 啊！因此， 尝试按下 [crtl]+q 应该就可以让整个画面重新恢复正常咯！

除了 stty 之外，其实我们的 bash 还有自己的一些终端机设置值呢！那就是利用 set 来设置的！ 我们之前提到一些变量时，可以利用 set 来显示，除此之外，其实 set 还可以帮我们设置整个指令输出/输入的环境。 例如记录历史命令、显示错误内容等等。

```
[dmtsai@study ~]$ set [-uvCHhmBx]
选项与参数：
-u  ：默认不启用。若启用后，当使用未设置变量时，会显示错误讯息；
-v  ：默认不启用。若启用后，在讯息被输出前，会先显示讯息的原始内容；
-x  ：默认不启用。若启用后，在指令被执行前，会显示指令内容（前面有 ++ 符号）
-h  ：默认启用。与历史命令有关；
-H  ：默认启用。与历史命令有关；
-m  ：默认启用。与工作管理有关；
-B  ：默认启用。与刮号 [] 的作用有关；
-C  ：默认不启用。若使用 &gt; 等，则若文件存在时，该文件不会被覆盖。

范例一：显示目前所有的 set 设置值
[dmtsai@study ~]$ echo $-
himBH
# 那个 $- 变量内容就是 set 的所有设置啦！ bash 默认是 himBH 喔！

范例二：设置 "若使用未定义变量时，则显示错误讯息"
[dmtsai@study ~]$ set -u
[dmtsai@study ~]$ echo $vbirding
-bash: vbirding: unbound variable
# 默认情况下，未设置/未宣告 的变量都会是“空的”，不过，若设置 -u 参数，
# 那么当使用未设置的变量时，就会有问题啦！很多的 shell 都默认启用 -u 参数。
# 若要取消这个参数，输入 set +u 即可！

范例三：执行前，显示该指令内容。
[dmtsai@study ~]$ set -x
++ printf '\033]0;%s@%s:%s\007' dmtsai study '~'    # 这个是在列出提示字符的控制码！
[dmtsai@study ~]$ echo ${HOME}
+ echo /home/dmtsai
/home/dmtsai
++ printf '\033]0;%s@%s:%s\007' dmtsai study '~'
# 看见否？要输出的指令都会先被打印到屏幕上喔！前面会多出 + 的符号！
```

另外，其实我们还有其他的按键设置功能呢！就是在前一小节提到的 /etc/inputrc 这个文件里面设置。 还有例如 /etc/DIRCOLORS 与 /usr/share/terminfo/ 等，也都是与终端机有关的环境设置文件呢！ 不过，事实上，鸟哥并不建议您修改 *tty* 的环境呢，这是因为 *bash* 的环境已经设置的很友好了， 我们不需要额外的设置或者修改，否则反而会产生一些困扰。不过，写在这里的数据， 只是希望大家能够清楚的知道我们的终端机是如何进行设置的喔！ ^^！ 最后，我们将 bash 默认的组合键给他汇整如下：

组合按键	执行结果
Ctrl + C	终止目前的命令
Ctrl + D	输入结束 （EOF），例如邮件结束的时候；
Ctrl + M	就是 Enter 啦！
Ctrl + S	暂停屏幕的输出
Ctrl + Q	恢复屏幕的输出
Ctrl + U	在提示字符下，将整列命令删除
Ctrl + Z	“暂停”目前的命令

10.4.5 万用字符与特殊符号

在 bash 的操作环境中还有一个非常有用的功能，那就是万用字符 （wildcard） ！ 我们利用 bash 处理数据就更方便了！下面我们列出一些常用的万用字符喔：

符号	意义
*	代表“ 0 个到无穷多个”任意字符
?	代表“一定有一个”任意字符
[ ]	同样代表“一定有一个在括号内”的字符（非任意字符）。例如 [abcd] 代表“一定有一个字符， 可能是 a, b, c, d 这四个任何一个”
[-]	若有减号在中括号内时，代表“在编码顺序内的所有字符”。例如 [0-9] 代表 0 到 9 之间的所有数字，因为数字的语系编码是连续的！
[^]	若中括号内的第一个字符为指数符号 （^） ，那表示“反向选择”，例如 [^abc] 代表一定有一个字符，只要是非 a, b, c 的其他字符就接受的意思。

接下来让我们利用万用字符来玩些东西吧！首先，利用万用字符配合 ls 找文件名看看：

```
[dmtsai@study ~]$ LANG=C              &lt;==由于与编码有关，先设置语系一下

范例一：找出 /etc/ 下面以 cron 为开头的文件名
[dmtsai@study ~]$ ll -d /etc/cron*    &lt;==加上 -d 是为了仅显示目录而已

范例二：找出 /etc/ 下面文件名“刚好是五个字母”的文件名
[dmtsai@study ~]$ ll -d /etc/?????    &lt;==由于 ? 一定有一个，所以五个 ? 就对了

范例三：找出 /etc/ 下面文件名含有数字的文件名
[dmtsai@study ~]$ ll -d /etc/*[0-9]*  &lt;==记得中括号左右两边均需 *

范例四：找出 /etc/ 下面，文件名开头非为小写字母的文件名：
[dmtsai@study ~]$ ll -d /etc/[^a-z]*  &lt;==注意中括号左边没有 *

范例五：将范例四找到的文件复制到 /tmp/upper 中
[dmtsai@study ~]$ mkdir /tmp/upper; cp -a /etc/[^a-z]* /tmp/upper
```

除了万用字符之外，bash 环境中的特殊符号有哪些呢？下面我们先汇整一下：

符号	内容
#	注解符号：这个最常被使用在 script 当中，视为说明！在后的数据均不执行
\	跳脱符号：将“特殊字符或万用字符”还原成一般字符
|	管线（pipe）：分隔两个管线命令的界定（后两节介绍）；
;	连续指令下达分隔符号：连续性命令的界定（注意！与管线命令并不相同）
~	使用者的主文件夹
$	取用变量前置字符：亦即是变量之前需要加的变量取代值
&	工作控制（job control）：将指令变成背景下工作
!	逻辑运算意义上的“非” not 的意思！
/	目录符号：路径分隔的符号
>, >>	数据流重导向：输出导向，分别是“取代”与“累加”
<, <<	数据流重导向：输入导向（这两个留待下节介绍）
' '	单引号，不具有变量置换的功能（$ 变为纯文本）
" "	具有变量置换的功能！（$ 可保留相关功能）
	两个“ ` ”中间为可以先执行的指令，亦可使用 $（ ）
（ ）	在中间为子 shell 的起始与结束
{ }	在中间为命令区块的组合！

以上为 bash 环境中常见的特殊符号汇整！理论上，你的“文件名”尽量不要使用到上述的字符啦！

10.5 数据流重导向

数据流重导向（redirect）由字面上的意思来看，好像就是将“数据给他传导到其他地方去”的样子？没错～数据流重导向就是将某个指令执行后应该要出现在屏幕上的数据，给他传输到其他的地方，例如文件或者是设备（例如打印机之类的）！这玩意儿在 Linux 的文字模式下面可重要的！尤其是如果我们想要将某些数据储存下来时，就更有用了！

10.5.1 什么是数据流重导向

什么是数据流重导向啊？这得要由指令的执行结果谈起！一般来说，如果你要执行一个指令，通常他会是这样的：

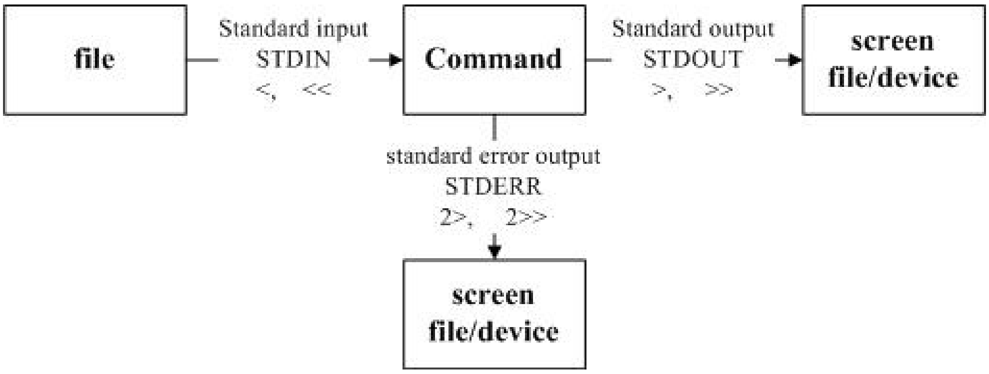


图10.5.1、指令执行过程的数据传输情况

我们执行一个指令的时候，这个指令可能会由文件读入数据，经过处理之后，再将数据输出到屏幕上。在上图当中，standard output 与 standard error output 分别代表“标准输出（STDOUT）”与“标准错误输出（STDERR）”，这两个玩意儿默认都是输出到屏幕上面来的啊！那么什么是标准输出与标准错误输出呢？

- standard output 与 standard error output

简单的说，标准输出指的是“指令执行所回传的正确的讯息”，而标准错误输出可理解为“ 指令执行失败后，所回传的错误讯息”。举个简单例子来说，我们的系统默认有 /etc/crontab 但却无 /etc/vbirdsay，此时若下达“ cat /etc/crontab /etc/vbirdsay ”这个指令时，cat 会进行：

- 标准输出：读取 /etc/crontab 后，将该文件内容显示到屏幕上；
- 标准错误输出：因为无法找到 /etc/vbirdsay，因此在屏幕上显示错误讯息

不管正确或错误的数据都是默认输出到屏幕上，所以屏幕当然是乱乱的！那能不能通过某些机制将这两股数据分开呢？当然可以啊！那就是数据流重导向的功能啊！数据流重导向可以将 standard output（简称 stdout）与 standard error output（简称 stderr）分别传送到其他的文件或设备去，而分别传送所用的特殊字符则如下所示：

1. 标准输入　（stdin）：代码为 0 ，使用 < 或 << ；

2. 标准输出　　（stdout）：代码为 1 ，使用 > 或 >> ；
3. 标准错误输出（stderr）：代码为 2 ，使用 2> 或 2>> ；

为了理解 stdout 与 stderr ，我们先来进行一个范例的练习：

```
范例一：观察你的系统根目录 （/） 下各目录的文件名、权限与属性，并记录下来
[dmtsai@study ~]$ ll /  &lt;==此时屏幕会显示出文件名信息

[dmtsai@study ~]$ ll / &gt; ~/rootfile &lt;==屏幕并无任何信息
[dmtsai@study ~]$ ll  ~/rootfile &lt;==有个新文件被创建了！
-rw-rw-r--. 1 dmtsai dmtsai 1078 Jul  9 18:51 /home/dmtsai/rootfile
```

怪了！屏幕怎么会完全没有数据呢？这是因为原本“ ll / ”所显示的数据已经被重新导向到 ~/rootfile 文件中了！ 那个 ~/rootfile 的文件名可以随便你取。如果你下达“ cat ~/rootfile ”那就可以看到原本应该在屏幕上面的数据啰。 如果我再次下达：“ ll /home > ~/rootfile ”后，那个 ~/rootfile 文件的内容变成什么？ 他将变成“仅有 ll /home 的数据”而已！咦！原本的“ ll / ”数据就不见了吗？是的！因为该文件的创建方式是：

1. 该文件 （本例中是 ~/rootfile） 若不存在，系统会自动的将他创建起来，但是
2. 当这个文件存在的时候，那么系统就会先将这个文件内容清空，然后再将数据写入！
3. 也就是若以 > 输出到一个已存在的文件中，那个文件就会被覆盖掉啰！

那如果我想要将数据累加而不想要将旧的数据删除，那该如何是好？利用两个大于的符号 （>>） 就好啦！以上面的范例来说，你应该要改成“ ll / >> ~/rootfile ”即可。 如此一来，当 （1） ~/rootfile 不存在时系统会主动创建这个文件；（2）若该文件已存在， 则数据会在该文件的最下方累加进去！

上面谈到的是 standard output 的正确数据，那如果是 standard error output 的错误数据呢？那就通过 2> 及 2>> 啰！同样是覆盖 （2>） 与累加 （2>>） 的特性！我们在刚刚才谈到 stdout 代码是 1 而 stderr 代码是 2 ， 所以这个 2> 是很容易理解的，而如果仅存在 > 时，则代表默认的代码 1 啰！也就是说：

- 1> ：以覆盖的方法将“正确的数据”输出到指定的文件或设备上；
- 1>>：以累加的方法将“正确的数据”输出到指定的文件或设备上；
- 2> ：以覆盖的方法将“错误的数据”输出到指定的文件或设备上；
- 2>>：以累加的方法将“错误的数据”输出到指定的文件或设备上；

要注意喔，“ 1>> ”以及“ 2>> ”中间是没有空格的！OK！有些概念之后让我们继续聊一聊这家伙怎么应用吧！ 当你以一般身份执行 find 这个指令的时候，由于权限的问题可能会产生一些错误信息。例如执行“ find / -name testing ”时，可能会产生类似“ find: /root: Permission denied ”之类的讯息。 例如下面这个范例：

```
范例二：利用一般身份帐号搜寻 /home 下面是否有名为 .bashrc 的文件存在
[dmtsai@study ~]$ find /home -name .bashrc &lt;==身份是 dmtsai 喔！
find: '/home/arod': Permission denied    &lt;== Standard error output
find: '/home/alex': Permission denied    &lt;== Standard error output
/home/dmtsai/.bashrc                     &lt;== Standard output
```

由于 /home 下面还有我们之前创建的帐号存在，那些帐号的主文件夹你当然不能进入啊！所以就会有错误及正确数据了。 好了，那么假如我想要将数据输出到 list 这个文件中呢？执行“ find /home -name .bashrc > list ” 会有什么结果？呵呵，你会发现 list 里面存了刚刚那个“正确”的输出数据， 至于屏幕上还是会有错误的讯息出现呢！伤脑筋！如果想要将正确的与错误的数据分别存入不同的文件中需要怎么做？

```
范例三：承范例二，将 stdout 与 stderr 分存到不同的文件去
[dmtsai@study ~]$ find /home -name .bashrc &gt; list_right 2&gt; list_error
```

注意喔，此时“屏幕上不会出现任何讯息”！因为刚刚执行的结果中，有 Permission 的那几行错误信息都会跑到 list_error 这个文件中，至于正确的输出数据则会存到 list_right 这个文件中啰！这样可以了解了吗？ 如果有点混乱的话，去休息一下再回来看看吧！

- /dev/null 垃圾桶黑洞设备与特殊写法

想像一下，如果我知道错误讯息会发生，所以要将错误讯息忽略掉而不显示或储存呢？ 这个时候黑洞设备 /dev/null 就很重要了！这个 /dev/null 可以吃掉任何导向这个设备的信息喔！将上述的范例修订一下：

```
范例四：承范例三，将错误的数据丢弃，屏幕上显示正确的数据
[dmtsai@study ~]$ find /home -name .bashrc 2&gt; /dev/null
/home/dmtsai/.bashrc  &lt;==只有 stdout 会显示到屏幕上， stderr 被丢弃了
```

再想像一下，如果我要将正确与错误数据通通写入同一个文件去呢？这个时候就得要使用特殊的写法了！ 我们同样用下面的案例来说明：

```
范例五：将指令的数据全部写入名为 list 的文件中
[dmtsai@study ~]$ find /home -name .bashrc &gt; list 2&gt; list  &lt;==错误
[dmtsai@study ~]$ find /home -name .bashrc &gt; list 2&gt;&1     &lt;==正确
[dmtsai@study ~]$ find /home -name .bashrc &&gt; list         &lt;==正确
```

上述表格第一行错误的原因是，由于两股数据同时写入一个文件，又没有使用特殊的语法，此时两股数据可能会交叉写入该文件内，造成次序的错乱。所以虽然最终 list 文件还是会产生，但是里面的数据排列就会怪怪的，而不是原本屏幕上的输出排序。 至于写入同一个文件的特殊语法如上表所示，你可以使用 2>&1 也可以使用 &> ！ 一般来说，鸟哥比较习惯使用 2>&1 的语法啦！

- standard input ： < 与 <<

了解了 stderr 与 stdout 后，那么那个 < 又是什么呀？呵呵！以最简单的说法来说， 那就是“将原本需要由键盘输入的数据，改由文件内容来取代”的意思。 我们先由下面的 cat 指令操作来了解一下什么叫做“键盘输入”吧！

```
范例六：利用 cat 指令来创建一个文件的简单流程
[dmtsai@study ~]$ cat &gt; catfile
testing
cat file test
&lt;==这里按下 [ctrl]+d 来离开

[dmtsai@study ~]$ cat catfile
testing
cat file test
```

由于加入 > 在 cat 后，所以那个 catfile 会被主动的创建，而内容就是刚刚键盘上面输入的那两行数据了。 唔！那我能不能用纯文本文件取代键盘的输入，也就是说，用某个文件的内容来取代键盘的敲击呢？ 可以的！如下所示：

```
范例七：用 stdin 取代键盘的输入以创建新文件的简单流程
[dmtsai@study ~]$ cat &gt; catfile &lt; ~/.bashrc
[dmtsai@study ~]$ ll catfile ~/.bashrc
-rw-r--r--. 1 dmtsai dmtsai 231 Mar  6 06:06 /home/dmtsai/.bashrc
-rw-rw-r--. 1 dmtsai dmtsai 231 Jul  9 18:58 catfile
# 注意看，这两个文件的大小会一模一样！几乎像是使用 cp 来复制一般！
```

这东西非常的有帮助！尤其是用在类似 mail 这种指令的使用上。 理解 < 之后，再来则是怪可怕一把的 << 这个连续两个小于的符号了。 他代表的是“结束的输入字符”的意思！举例来讲：“我要用 cat 直接将输入的讯息输出到 catfile 中， 且当由键盘输入 eof 时，该次输入就结束”，那我可以这样做：

```
[dmtsai@study ~]$ cat &gt; catfile &lt;&lt; "eof"
&gt; This is a test.
&gt; OK now stop
&gt; eof  &lt;==输入这关键字，立刻就结束而不需要输入 [ctrl]+d

[dmtsai@study ~]$ cat catfile
This is a test.
OK now stop     &lt;==只有这两行，不会存在关键字那一行！
```

看到了吗？利用 << 右侧的控制字符，我们可以终止一次输入， 而不必输入 [crtl]+d 来结束哩！这对程序写作很有帮助喔！好了，那么为何要使用命令输出重导向呢？我们来说一说吧！

- 屏幕输出的信息很重要，而且我们需要将他存下来的时候；
- 背景执行中的程序，不希望他干扰屏幕正常的输出结果时；
- 一些系统的例行命令 （例如写在 /etc/crontab 中的文件） 的执行结果，希望他可以存下来时；
- 一些执行命令的可能已知错误讯息时，想以“ 2> /dev/null ”将他丢掉时；
- 错误讯息与正确讯息需要分别输出时。

当然还有很多的功能的，最简单的就是网友们常常问到的：“为何我的 root 都会收到系统 crontab 寄来的错误讯息呢”这个咚咚是常见的错误， 而如果我们已经知道这个错误讯息是可以忽略的时候，嗯！“ 2> errorfile ”这个功能就很重要了吧！ 了解了吗？

问：假设我要将 echo "error message" 以 standard error output 的格式来输出，该如何处置？答：既然有 2>&1 来将 2> 转到 1> 去，那么应该也会有 1>&2 吧？没错！就是这个概念！因此你可以这样作：

```
[dmtsai@study ~]$ echo "error message" 1&gt;&2
[dmtsai@study ~]$ echo "error message" 2&gt; /dev/null 1&gt;&2
```

你会发现第一条有讯息输出到屏幕上，第二条则没有讯息！这表示该讯息已经是通过 2> /dev/null 丢到垃圾桶去了！ 可以肯定是错误讯息啰！ ^_^

10.5.2 命令执行的判断依据： ; , &&, ||

在某些情况下，很多指令我想要一次输入去执行，而不想要分次执行时，该如何是好？基本上你有两个选择， 一个是通过第十二章要介绍的 shell script 撰写脚本去执行，一种则是通过下面的介绍来一次输入多重指令喔！

- cmd ; cmd （不考虑指令相关性的连续指令下达）

在某些时候，我们希望可以一次执行多个指令，例如在关机的时候我希望可以先执行两次 sync 同步化写入磁盘后才 shutdown 计算机，那么可以怎么作呢？这样做呀：

```
[root@study ~]# sync; sync; shutdown -h now
```

在指令与指令中间利用分号 (;) 来隔开，这样一来，分号前的指令执行完后就会立刻接着执行后面的指令了。 这真是方便啊～再来，换个角度来想，万一我想要在某个目录下面创建一个文件，也就是说，如果该目录存在的话， 那我才创建这个文件，如果不存在，那就算了。也就是说这两个指令彼此之间是有相关性的， 前一个指令是否成功的执行与后一个指令是否要执行有关！那就得动用到 && 或 || 啰！

- $? (指令回传值) 与 && 或 ||

如同上面谈到的，两个指令之间有相依性，而这个相依性主要判断的地方就在于前一个指令执行的结果是否正确。 还记得本章之前我们曾介绍过指令回传值吧！嘿嘿！没错，您真聪明！就是通过这个回传值啦！ 再复习一次“若前一个指令执行的结果为正确，在 Linux 下面会回传一个 $? = 0 的值”。 那么我们怎么通过这个回传值来判断后续的指令是否要执行呢？这就得要借由“ && ”及“ || ”的帮忙了！ 注意喔，两个 & 之间是没有空格的！那个 | 则是 [Shift]+[] 的按键结果。

指令下达情况	说明
cmd1 && cmd2	1. 若 cmd1 执行完毕且正确执行（$?=0），则开始执行 cmd2。 2. 若 cmd1 执行完毕且为错误 （$?≠0），则 cmd2 不执行。
cmd1 || cmd2	1. 若 cmd1 执行完毕且正确执行（$?=0），则 cmd2 不执行。 2. 若 cmd1 执行完毕且为错误 （$?≠0），则开始执行 cmd2。

上述的 cmd1 及 cmd2 都是指令。好了，回到我们刚刚假想的情况，就是想要： （1）先判断一个目录是否存在； （2）若存在才在该目录下面创建一个文件。由于我们尚未介绍如何判断式 （test） 的使用，在这里我们使用 ls 以及回传值来判断目录是否存在啦！ 让我们进行下面这个练习看看：

```
范例一：使用 ls 查阅目录 /tmp/abc 是否存在，若存在则用 touch 创建 /tmp/abc/hehe
[dmtsai@study ~]$ ls /tmp/abc && touch /tmp/abc/hehe
ls: cannot access /tmp/abc: No such file or directory
# ls 很干脆的说明找不到该目录，但并没有 touch 的错误，表示 touch 并没有执行

[dmtsai@study ~]$ mkdir /tmp/abc
[dmtsai@study ~]$ ls /tmp/abc && touch /tmp/abc/hehe
[dmtsai@study ~]$ ll /tmp/abc
-rw-rw-r--. 1 dmtsai dmtsai 0 Jul  9 19:16 hehe
```

看到了吧？如果 /tmp/abc 不存在时，touch 就不会被执行，若 /tmp/abc 存在的话，那么 touch 就会开始执行啰！ 很不错用吧！不过，我们还得手动自行创建目录，伤脑筋～能不能自动判断，如果没有该目录就给予创建呢？ 参考一下下面的例子先：

```
范例二：测试 /tmp/abc 是否存在，若不存在则予以创建，若存在就不作任何事情
[dmtsai@study ~]$ rm -r /tmp/abc                &lt;==先删除此目录以方便测试
[dmtsai@study ~]$ ls /tmp/abc &#124;&#124; mkdir /tmp/abc
ls: cannot access /tmp/abc: No such file or directory  &lt;==真的不存在喔！
[dmtsai@study ~]$ ll -d /tmp/abc
drwxrwxr-x. 2 dmtsai dmtsai 6 Jul  9 19:17 /tmp/abca   &lt;==结果出现了！有进行 mkdir
```

如果你一再重复“ ls /tmp/abc || mkdir /tmp/abc ”画面也不会出现重复 mkdir 的错误！这是因为 /tmp/abc 已经存在， 所以后续的 mkdir 就不会进行！这样理解否？好了，让我们再次的讨论一下，如果我想要创建 /tmp/abc/hehe 这个文件， 但我并不知道 /tmp/abc 是否存在，那该如何是好？试看看：

```
范例三：我不清楚 /tmp/abc 是否存在，但就是要创建 /tmp/abc/hehe 文件
[dmtsai@study ~]$ ls /tmp/abc &#124;&#124; mkdir /tmp/abc && touch /tmp/abc/hehe
```

上面这个范例三总是会尝试创建 /tmp/abc/hehe 的喔！不论 /tmp/abc 是否存在。那么范例三应该如何解释呢？ 由于Linux 下面的指令都是由左往右执行的，所以范例三有几种结果我们来分析一下：

- （1）若 /tmp/abc 不存在故回传 $?≠0，则 （2）因为 || 遇到非为 0 的 $? 故开始 mkdir /tmp/abc，由于 mkdir /tmp/abc 会成功进行，所以回传 $?=0 （3）因为 && 遇到 $?=0 故会执行 touch /tmp/abc/hehe，最终 hehe 就被创建了；
- （1）若 /tmp/abc 存在故回传 $?=0，则 （2）因为 || 遇到 0 的 $? 不会进行，此时 $?=0 继续向后传，故 （3）因为 && 遇到 $?=0 就开始创建 /tmp/abc/hehe 了！最终 /tmp/abc/hehe 被创建起来。

整个流程图示如下：

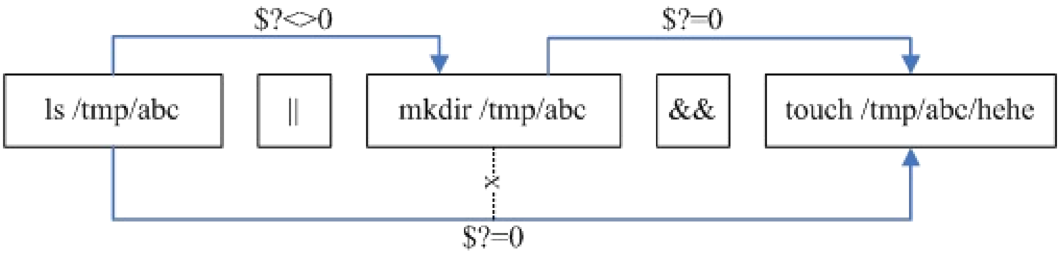


图10.5.2、指令依序执行的关系示意图

上面这张图显示的两股数据中，上方的线段为不存在 /tmp/abc 时所进行的指令行为，下方的线段则是存在 /tmp/abc 所在的指令行为。如上所述，下方线段由于存在 /tmp/abc 所以导致 $?=0 ，让中间的 mkdir 就不执行了！ 并将 $?=0 继续往后传给后续的 touch 去利用啦！瞭乎？在任何时刻你都可以拿上面这张图作为示意！ 让我们来想想下面这个例题吧！

例题：以 ls 测试 /tmp/vbirding 是否存在，若存在则显示 "exist" ，若不存在，则显示 "not exist"！答：这又牵涉到逻辑判断的问题，如果存在就显示某个数据，若不存在就显示其他数据，那我可以这样做：

> ls /tmp/vbirding && echo "exist" || echo "not exist"

意思是说，当 ls /tmp/vbirding 执行后，若正确，就执行 echo "exist" ，若有问题，就执行 echo "not exist" ！那如果写成如下的状况会出现什么？

> ls /tmp/vbirding || echo "not exist" && echo "exist"

这其实是有问题的，为什么呢？由图 10.5.2 的流程介绍我们知道指令是一个一个往后执行，因此在上面的例子当中，如果 /tmp/vbirding 不存在时，他会进行如下动作：

1. 若 ls /tmp/vbirding 不存在，因此回传一个非为 0 的数值；
2. 接下来经过 || 的判断，发现前一个指令回传非为 0 的数值，因此，程序开始执行 echo "not exist" ，而 echo "not exist" 程序肯定可以执行成功，因此会回传一个 0 值给后面的指令；
3. 经过 && 的判断，咦！是 0 啊！所以就开始执行 echo "exist" 。

所以啊，嘿嘿！第二个例子里面竟然会同时出现 not exist 与 exist 呢！真神奇～

经过这个例题的练习，你应该会了解，由于指令是一个接着一个去执行的，因此，如果真要使用判断， 那么这个 && 与 || 的顺序就不能搞错。一般来说，假设判断式有三个，也就是：

> command1 && command2 || command3

而且顺序通常不会变，因为一般来说， command2 与 command3 会放置肯定可以执行成功的指令， 因此，依据上面例题的逻辑分析，您就会晓得为何要如此放置啰～这很有用的啦！而且.....考试也很常考～

10.6 管线命令 （pipe）

就如同前面所说的， bash 命令执行的时候有输出的数据会出现！ 那么如果这群数据必需要经过几道手续之后才能得到我们所想要的格式，应该如何来设置？ 这就牵涉到管线命令的问题了 （pipe） ，管线命令使用的是“ | ”这个界定符号！ 另外，管线命令与“连续下达命令”是不一样的呦！ 这点下面我们会再说明。下面我们先举一个例子来说明一下简单的管线命令。

假设我们想要知道 /etc/ 下面有多少文件，那么可以利用 ls /etc 来查阅，不过， 因为 /etc 下面的文件太多，导致一口气就将屏幕塞满了～不知道前面输出的内容是啥？此时，我们可以通过 less 指令的协助，利用：

```
[dmtsai@study ~]$ ls -al /etc &#124; less
```

如此一来，使用 ls 指令输出后的内容，就能够被 less 读取，并且利用 less 的功能，我们就能够前后翻动相关的信息了！很方便是吧？我们就来了解一下这个管线命令“ | ”的用途吧！ 其实这个管线命令“ | ”仅能处理经由前面一个指令传来的正确信息，也就是 standard output 的信息，对于 stdandard error 并没有直接处理的能力。那么整体的管线命令可以使用下图表示：

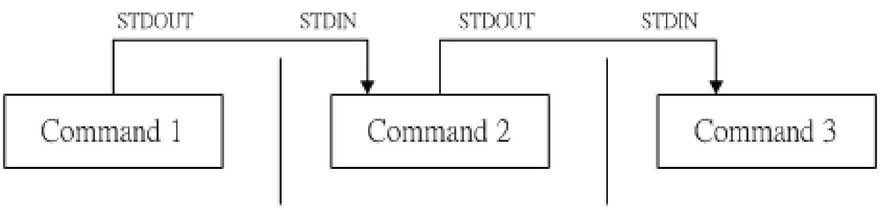


图10.6.1、管线命令的处理示意图

在每个管线后面接的第一个数据必定是“指令”喔！而且这个指令必须要能够接受 standard input 的数据才行，这样的指令才可以是为“管线命令”，例如 less, more, head, tail 等都是可以接受 standard input 的管线命令啦。至于例如 ls, cp, mv 等就不是管线命令了！因为 ls, cp, mv 并不会接受来自 stdin 的数据。 也就是说，管线命令主要有两个比较需要注意的地方：

- 管线命令仅会处理 standard output，对于 standard error output 会予以忽略
- 管线命令必须要能够接受来自前一个指令的数据成为 standard input 继续处理才行。



Tips 想一想，如果你硬要让 standard error 可以被管线命令所使用，那该如何处理？其实就是通过上一小节的数据流重导向即可！ 让 2>&1 加入指令中～就可以让 2> 变成 1> 啰！了解了吗？ ^_^

多说无益，让我们来玩一些管线命令吧！下面的咚咚对系统管理非常有帮助喔！

10.6.1 撷取命令： cut, grep

什么是撷取命令啊？说穿了，就是将一段数据经过分析后，取出我们所想要的。或者是经由分析关键字，取得我们所想要的那一行！ 不过，要注意的是，一般来说，撷取讯息通常是针对“一行一行”来分析的， 并不是整篇讯息分析的喔～下面我们介绍两个很常用的讯息撷取命令：

- cut

cut 不就是“切”吗？没错啦！这个指令可以将一段讯息的某一段给他“切”出来～ 处理的讯息是以“行”为单位喔！下面我们就来谈一谈：

```
[dmtsai@study ~]$ cut -d'分隔字符' -f fields &lt;==用于有特定分隔字符
[dmtsai@study ~]$ cut -c 字符区间            &lt;==用于排列整齐的讯息
选项与参数：
-d  ：后面接分隔字符。与 -f 一起使用；
-f  ：依据 -d 的分隔字符将一段讯息分区成为数段，用 -f 取出第几段的意思；
-c  ：以字符 （characters） 的单位取出固定字符区间；

范例一：将 PATH 变量取出，我要找出第五个路径。
[dmtsai@study ~]$ echo ${PATH}
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dmtsai/bi
#      1      &#124;    2   &#124;       3       &#124;    4    &#124;           5

[dmtsai@study ~]$ echo ${PATH} &#124; cut -d ':' -f 5
# 如同上面的数字显示，我们是以“ : ”作为分隔，因此会出现 /home/dmtsai/.local/bin
# 那么如果想要列出第 3 与第 5 呢？，就是这样：
[dmtsai@study ~]$ echo ${PATH} &#124; cut -d ':' -f 3,5

范例二：将 export 输出的讯息，取得第 12 字符以后的所有字串
[dmtsai@study ~]$ export
declare -x HISTCONTROL="ignoredups"
declare -x HISTSIZE="1000"
declare -x HOME="/home/dmtsai"
declare -x HOSTNAME="study.centos.vbird"
.....（其他省略）.....
# 注意看，每个数据都是排列整齐的输出！如果我们不想要“ declare -x ”时，就得这么做：

[dmtsai@study ~]$ export &#124; cut -c 12-
HISTCONTROL="ignoredups"
HISTSIZE="1000"
HOME="/home/dmtsai"
HOSTNAME="study.centos.vbird"
.....（其他省略）.....
# 知道怎么回事了吧？用 -c 可以处理比较具有格式的输出数据！
# 我们还可以指定某个范围的值，例如第 12-20 的字符，就是 cut -c 12-20 等等！

范例三：用 last 将显示的登陆者的信息中，仅留下使用者大名
[dmtsai@study ~]$ last
root   pts/1    192.168.201.101  Sat Feb  7 12:35   still logged in
root   pts/1    192.168.201.101  Fri Feb  6 12:13 - 18:46  （06:33）
root   pts/1    192.168.201.254  Thu Feb  5 22:37 - 23:53  （01:16）
# last 可以输出“帐号/终端机/来源/日期时间”的数据，并且是排列整齐的

[dmtsai@study ~]$ last &#124; cut -d ' ' -f 1
# 由输出的结果我们可以发现第一个空白分隔的字段代表帐号，所以使用如上指令：
# 但是因为 root   pts/1 之间空格有好几个，并非仅有一个，所以，如果要找出
# pts/1 其实不能以 cut -d ' ' -f 1,2 喔！输出的结果会不是我们想要的。
```

cut 主要的用途在于将“同一行里面的数据进行分解！”最常使用在分析一些数据或文字数据的时候！ 这是因为有时候我们会以某些字符当作分区的参数，然后来将数据加以切割，以取得我们所需要的数据。 鸟哥也很常使用这个功能呢！尤其是在分析 log 文件的时候！不过，cut 在处理多空格相连的数据时，可能会比较吃力一点，所以某些时刻可能会使用下一章的 awk 来取代的！

- grep

刚刚的 cut 是将一行讯息当中，取出某部分我们想要的，而 grep 则是分析一行讯息， 若当中有我们所需要的信息，就将该行拿出来～简单的语法是这样的：

```
[dmtsai@study ~]$ grep [-acinv] [--color=auto] '搜寻字串' filename
选项与参数：
-a ：将 binary 文件以 text 文件的方式搜寻数据
-c ：计算找到 '搜寻字串' 的次数
-i ：忽略大小写的不同，所以大小写视为相同
-n ：顺便输出行号
-v ：反向选择，亦即显示出没有 '搜寻字串' 内容的那一行！
--color=auto ：可以将找到的关键字部分加上颜色的显示喔！

范例一：将 last 当中，有出现 root 的那一行就取出来；
[dmtsai@study ~]$ last &#124; grep 'root'

范例二：与范例一相反，只要没有 root 的就取出！
[dmtsai@study ~]$ last &#124; grep -v 'root'

范例三：在 last 的输出讯息中，只要有 root 就取出，并且仅取第一栏
[dmtsai@study ~]$ last &#124; grep 'root' &#124;cut -d ' ' -f1
# 在取出 root 之后，利用上个指令 cut 的处理，就能够仅取得第一栏啰！

范例四：取出 /etc/man_db.conf 内含 MANPATH 的那几行
[dmtsai@study ~]$ grep --color=auto 'MANPATH' /etc/man_db.conf
....（前面省略）....
MANPATH_MAP     /usr/games              /usr/share/man
MANPATH_MAP     /opt/bin                /opt/man
MANPATH_MAP     /opt/sbin               /opt/man
# 神奇的是，如果加上 --color=auto 的选项，找到的关键字部分会用特殊颜色显示喔！
```

grep 是个很棒的指令喔！他支持的语法实在是太多了～用在正则表达式里头， 能够处理的数据实在是多的很～不过，我们这里先不谈正则表达式～下一章再来说明～ 您先了解一下， grep 可以解析一行文字，取得关键字，若该行有存在关键字，就会整行列出来！另外， CentOS 7 当中，默认的 grep 已经主动加上 --color=auto 在 alias 内了喔！

10.6.2 排序命令： sort, wc, uniq

很多时候，我们都会去计算一次数据里头的相同型态的数据总数，举例来说， 使用 last 可以查得系统上面有登陆主机者的身份。那么我可以针对每个使用者查出他们的总登陆次数吗？此时就得要排序与计算之类的指令来辅助了！下面我们介绍几个好用的排序与统计指令喔！

- sort

sort 是很有趣的指令，他可以帮我们进行排序，而且可以依据不同的数据型态来排序喔！ 例如数字与文字的排序就不一样。此外，排序的字符与语系的编码有关，因此， 如果您需要排序时，建议使用 LANG=C 来让语系统一，数据排序比较好一些。

```
[dmtsai@study ~]$ sort [-fbMnrtuk] [file or stdin]
选项与参数：
-f  ：忽略大小写的差异，例如 A 与 a 视为编码相同；
-b  ：忽略最前面的空白字符部分；
-M  ：以月份的名字来排序，例如 JAN, DEC 等等的排序方法；
-n  ：使用“纯数字”进行排序（默认是以文字体态来排序的）；
-r  ：反向排序；
-u  ：就是 uniq ，相同的数据中，仅出现一行代表；
-t  ：分隔符号，默认是用 [tab] 键来分隔；
-k  ：以那个区间 （field） 来进行排序的意思

范例一：个人帐号都记录在 /etc/passwd 下，请将帐号进行排序。
[dmtsai@study ~]$ cat /etc/passwd &#124; sort
abrt:x:173:173::/etc/abrt:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
alex:x:1001:1002::/home/alex:/bin/bash
# 鸟哥省略很多的输出～由上面的数据看起来， sort 是默认“以第一个”数据来排序，
# 而且默认是以“文字”型态来排序的喔！所以由 a 开始排到最后啰！

范例二：/etc/passwd 内容是以 : 来分隔的，我想以第三栏来排序，该如何？
[dmtsai@study ~]$ cat /etc/passwd &#124; sort -t ':' -k 3
root:x:0:0:root:/root:/bin/bash
dmtsai:x:1000:1000:dmtsai:/home/dmtsai:/bin/bash
alex:x:1001:1002::/home/alex:/bin/bash
arod:x:1002:1003::/home/arod:/bin/bash
# 看到特殊字体的输出部分了吧？怎么会这样排列啊？呵呵！没错啦～
# 如果是以文字体态来排序的话，原本就会是这样，想要使用数字排序：
# cat /etc/passwd &#124; sort -t ':' -k 3 -n
# 这样才行啊！用那个 -n 来告知 sort 以数字来排序啊！

范例三：利用 last ，将输出的数据仅取帐号，并加以排序
[dmtsai@study ~]$ last &#124; cut -d ' ' -f1 &#124; sort
```

sort 同样是很常用的指令呢！因为我们常常需要比较一些信息啦！ 举个上面的第二个例子来说好了！今天假设你有很多的帐号，而且你想要知道最大的使用者 ID 目前到哪一号了！呵呵！使用 sort 一下子就可以知道答案咯！当然其使用还不止此啦！有空的话不妨玩一玩！

- uniq

如果我排序完成了，想要将重复的数据仅列出一个显示，可以怎么做呢？

```
[dmtsai@study ~]$ uniq [-ic]
选项与参数：
-i   ：忽略大小写字符的不同；
-c   ：进行计数

范例一：使用 last 将帐号列出，仅取出帐号栏，进行排序后仅取出一位；
[dmtsai@study ~]$ last &#124; cut -d ' ' -f1 &#124; sort &#124; uniq

范例二：承上题，如果我还想要知道每个人的登陆总次数呢？
[dmtsai@study ~]$ last &#124; cut -d ' ' -f1 &#124; sort &#124; uniq -c
      1
      6 （unknown
     47 dmtsai
      4 reboot
      7 root
      1 wtmp
# 从上面的结果可以发现 reboot 有 4 次， root 登陆则有 7 次！大部分是以 dmtsai 来操作！
# wtmp 与第一行的空白都是 last 的默认字符，那两个可以忽略的！
```

这个指令用来将“重复的行删除掉只显示一个”，举个例子来说， 你要知道这个月份登陆你主机的使用者有谁，而不在乎他的登陆次数，那么就使用上面的范例， （1）先将所有的数据列出；（2）再将人名独立出来；（3）经过排序；（4）只显示一个！ 由于这个指令是在将重复的东西减少，所以当然需要“配合排序过的文件”来处理啰！

- wc

如果我想要知道 /etc/man_db.conf 这个文件里面有多少字？多少行？多少字符的话， 可以怎么做呢？其实可以利用 wc 这个指令来达成喔！他可以帮我们计算输出的讯息的整体数据！

```
[dmtsai@study ~]$ wc [-lwm]
选项与参数：
-l   ：仅列出行；
-w   ：仅列出多少字（英文单字）；
-m   ：多少字符；

范例一：那个 /etc/man_db.conf 里面到底有多少相关字、行、字符数？
[dmtsai@study ~]$ cat /etc/man_db.conf &#124; wc
    131     723    5171
# 输出的三个数字中，分别代表： “行、字数、字符数”

范例二：我知道使用 last 可以输出登陆者，但是 last 最后两行并非帐号内容，那么请问，
        我该如何以一行指令串取得登陆系统的总人次？
[dmtsai@study ~]$ last &#124; grep [a-zA-Z] &#124; grep -v 'wtmp' &#124; grep -v 'reboot'
&gt; grep -v 'unknown' &#124;wc -l
# 由于 last 会输出空白行, wtmp, unknown, reboot 等无关帐号登陆的信息，因此，我利用
# grep 取出非空白行，以及去除上述关键字那几行，再计算行数，就能够了解啰！
```

wc 也可以当作指令？这可不是上洗手间的 WC 呢！这是相当有用的计算文件内容的一个工具组喔！举个例子来说， 当你要知道目前你的帐号文件中有多少个帐号时，就使用这个方法：“ cat /etc/passwd | wc -l ”啦！因为 /etc/passwd 里头一行代表一个使用者呀！ 所以知道行数就晓得有多少的帐号在里头了！而如果要计算一个文件里头有多少个字符时，就使用 wc -m 这个选项吧！

10.6.3 双向重导向： tee

想个简单的东西，我们由前一节知道 > 会将数据流整个传送给文件或设备，因此我们除非去读取该文件或设备， 否则就无法继续利用这个数据流。万一我想要将这个数据流的处理过程中将某段讯息存下来，应该怎么做？ 利用 tee 就可以啰～我们可以这样简单的看一下：

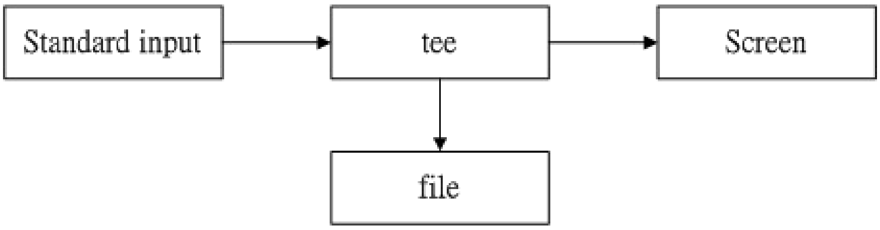


图10.6.2、tee 的工作流程示意图

tee 会同时将数据流分送到文件去与屏幕 （screen） ；而输出到屏幕的，其实就是 stdout ，那就可以让下个指令继续处理喔！

```
[dmtsai@study ~]$ tee [-a] file
选项与参数：
-a  ：以累加 （append） 的方式，将数据加入 file 当中！

[dmtsai@study ~]$ last &#124; tee last.list &#124; cut -d " " -f1
# 这个范例可以让我们将 last 的输出存一份到 last.list 文件中；

[dmtsai@study ~]$ ls -l /home &#124; tee ~/homefile &#124; more
# 这个范例则是将 ls 的数据存一份到 ~/homefile ，同时屏幕也有输出讯息！

[dmtsai@study ~]$ ls -l / &#124; tee -a ~/homefile &#124; more
# 要注意！ tee 后接的文件会被覆盖，若加上 -a 这个选项则能将讯息累加。
```

tee 可以让 standard output 转存一份到文件内并将同样的数据继续送到屏幕去处理！ 这样除了可以让我们同时分析一份数据并记录下来之外，还可以作为处理一份数据的中间暂存盘记录之用！ tee 这家伙在很多选择/填充的认证考试中很容易考呢！

10.6.4 字符转换命令： tr, col, join, paste, expand

我们在 vim 程序编辑器当中，提到过 DOS 断行字符与 Unix 断行字符的不同，并且可以使用 dos2unix 与 unix2dos 来完成转换。好了，那么思考一下，是否还有其他常用的字符替代？ 举例来说，要将大写改成小写，或者是将数据中的 [tab] 按键转成空白键？还有，如何将两篇讯息整合成一篇？ 下面我们就来介绍一下这些字符转换命令在管线当中的使用方法：

- tr

tr 可以用来删除一段讯息当中的文字，或者是进行文字讯息的替换！

```
[dmtsai@study ~]$ tr [-ds] SET1 ...
选项与参数：
-d  ：删除讯息当中的 SET1 这个字串；
-s  ：取代掉重复的字符！

范例一：将 last 输出的讯息中，所有的小写变成大写字符：
[dmtsai@study ~]$ last &#124; tr '[a-z]' '[A-Z]'
# 事实上，没有加上单引号也是可以执行的，如：“ last &#124; tr [a-z] [A-Z] ”

范例二：将 /etc/passwd 输出的讯息中，将冒号 (:) 删除
[dmtsai@study ~]$ cat /etc/passwd &#124; tr -d ':'

范例三：将 /etc/passwd 转存成 dos 断行到 /root/passwd 中，再将 ^M 符号删除
[dmtsai@study ~]$ cp /etc/passwd ~/passwd && unix2dos ~/passwd
[dmtsai@study ~]$ file /etc/passwd ~/passwd
/etc/passwd:         ASCII text
/home/dmtsai/passwd: ASCII text, with CRLF line terminators  &lt;==就是 DOS 断行
[dmtsai@study ~]$ cat ~/passwd &#124; tr -d '\r' &gt; ~/passwd.linux
# 那个 \r 指的是 DOS 的断行字符，关于更多的字符，请参考 man tr
[dmtsai@study ~]$ ll /etc/passwd ~/passwd*
-rw-r--r--. 1 root   root   2092 Jun 17 00:20 /etc/passwd
-rw-r--r--. 1 dmtsai dmtsai 2133 Jul  9 22:13 /home/dmtsai/passwd
-rw-rw-r--. 1 dmtsai dmtsai 2092 Jul  9 22:13 /home/dmtsai/passwd.linux
# 处理过后，发现文件大小与原本的 /etc/passwd 就一致了！
```

其实这个指令也可以写在“正则表达式”里头！因为他也是由正则表达式的方式来取代数据的！以上面的例子来说，使用 [] 可以设置一串字呢！也常常用来取代文件中的怪异符号！ 例如上面第三个例子当中，可以去除 DOS 文件留下来的 ^M 这个断行的符号！这东西相当的有用！相信处理 Linux & Windows 系统中的人们最麻烦的一件事就是这个事情啦！亦即是 DOS 下面会自动的在每行行尾加入 ^M 这个断行符号！这个时候除了以前讲过的 dos2unix 之外，我们也可以使用这个 tr 来将 ^M 去除！ ^M 可以使用 \r 来代替之！

- col

```
[dmtsai@study ~]$ col [-xb]
选项与参数：
-x  ：将 tab 键转换成对等的空白键

范例一：利用 cat -A 显示出所有特殊按键，最后以 col 将 [tab] 转成空白
[dmtsai@study ~]$ cat -A /etc/man_db.conf  &lt;==此时会看到很多 ^I 的符号，那就是 tab
[dmtsai@study ~]$ cat /etc/man_db.conf &#124; col -x &#124; cat -A &#124; more
# 嘿嘿！如此一来， [tab] 按键会被取代成为空白键，输出就美观多了！
```

虽然 col 有他特殊的用途，不过，很多时候，他可以用来简单的处理将 [tab] 按键取代成为空白键！ 例如上面的例子当中，如果使用 cat -A 则 [tab] 会以 ^I 来表示。 但经过 col -x 的处理，则会将 [tab] 取代成为对等的空白键！

- join

join 看字面上的意义 （加入/参加） 就可以知道，他是在处理两个文件之间的数据， 而且，主要是在处理“两个文件当中，有 "相同数据" 的那一行，才将他加在一起”的意思。我们利用下面的简单例子来说明：

```
[dmtsai@study ~]$ join [-ti12] file1 file2
选项与参数：
-t  ：join 默认以空白字符分隔数据，并且比对“第一个字段”的数据，
      如果两个文件相同，则将两笔数据联成一行，且第一个字段放在第一个！
-i  ：忽略大小写的差异；
-1  ：这个是数字的 1 ，代表“第一个文件要用那个字段来分析”的意思；
-2  ：代表“第二个文件要用那个字段来分析”的意思。

范例一：用 root 的身份，将 /etc/passwd 与 /etc/shadow 相关数据整合成一栏
[root@study ~]# head -n 3 /etc/passwd /etc/shadow
==&gt; /etc/passwd &lt;==
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin

==&gt; /etc/shadow &lt;==
root:$6$wtbCCce/PxMeE5wm$KE2IfSJr...:16559:0:99999:7:::
bin:*:16372:0:99999:7:::
daemon:*:16372:0:99999:7:::
# 由输出的数据可以发现这两个文件的最左边字段都是相同帐号！且以 : 分隔

[root@study ~]# join -t ':' /etc/passwd /etc/shadow &#124; head -n 3
root:x:0:0:root:/root:/bin/bash:$6$wtbCCce/PxMeE5wm$KE2IfSJr...:16559:0:99999:7:::
bin:x:1:1:bin:/bin:/sbin/nologin:*:16372:0:99999:7:::
daemon:x:2:2:daemon:/sbin:/sbin/nologin:*:16372:0:99999:7:::
# 通过上面这个动作，我们可以将两个文件第一字段相同者整合成一列！
# 第二个文件的相同字段并不会显示（因为已经在最左边的字段出现了啊！）

范例二：我们知道 /etc/passwd 第四个字段是 GID ，那个 GID 记录在
        /etc/group 当中的第三个字段，请问如何将两个文件整合？
[root@study ~]# head -n 3 /etc/passwd /etc/group
==&gt; /etc/passwd &lt;==
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin

==&gt; /etc/group &lt;==
root:x:0:
bin:x:1:
daemon:x:2:
# 从上面可以看到，确实有相同的部分喔！赶紧来整合一下！

[root@study ~]# join -t ':' -1 4 /etc/passwd -2 3 /etc/group &#124; head -n 3
0:root:x:0:root:/root:/bin/bash:root:x:
1:bin:x:1:bin:/bin:/sbin/nologin:bin:x:
2:daemon:x:2:daemon:/sbin:/sbin/nologin:daemon:x:
# 同样的，相同的字段部分被移动到最前面了！所以第二个文件的内容就没再显示。
# 请读者们配合上述显示两个文件的实际内容来比对！
```

这个 join 在处理两个相关的数据文件时，就眞的是很有帮助的啦！ 例如上面的案例当中，我的 /etc/passwd, /etc/shadow, /etc/group 都是有相关性的， 其中 /etc/passwd, /etc/shadow 以帐号为相关性，至于 /etc/passwd, /etc/group 则以所谓的 GID （帐号的数字定义） 来作为他的相关性。根据这个相关性， 我们可以将有关系的数据放置在一起！这在处理数据可是相当有帮助的！ 但是上面的例子有点难，希望您可以静下心好好的看一看原因喔！

此外，需要特别注意的是，在使用 join 之前，你所需要处理的文件应该要事先经过排序（sort） 处理！ 否则有些比对的项目会被略过呢！特别注意了！

- paste

这个 paste 就要比 join 简单多了！相对于 join 必须要比对两个文件的数据相关性， paste 就直接“将两行贴在一起，且中间以 [tab] 键隔开”而已！简单的使用方法：

```
[dmtsai@study ~]$ paste [-d] file1 file2
选项与参数：
-d  ：后面可以接分隔字符。默认是以 [tab] 来分隔的！
-   ：如果 file 部分写成 - ，表示来自 standard input 的数据的意思。

范例一：用 root 身份，将 /etc/passwd 与 /etc/shadow 同一行贴在一起
[root@study ~]# paste /etc/passwd /etc/shadow
root:x:0:0:root:/root:/bin/bash root:$6$wtbCCce/PxMeE5wm$KE2IfSJr...:16559:0:99999:7:::
bin:x:1:1:bin:/bin:/sbin/nologin        bin:*:16372:0:99999:7:::
daemon:x:2:2:daemon:/sbin:/sbin/nologin daemon:*:16372:0:99999:7:::
# 注意喔！同一行中间是以 [tab] 按键隔开的！

范例二：先将 /etc/group 读出（用 cat），然后与范例一贴上一起！且仅取出前三行
[root@study ~]# cat /etc/group&#124;paste /etc/passwd /etc/shadow -&#124;head -n 3
# 这个例子的重点在那个 - 的使用！那玩意儿常常代表 stdin 喔！
```

- expand

这玩意儿就是在将 [tab] 按键转成空白键啦～可以这样玩：

```
[dmtsai@study ~]$ expand [-t] file
选项与参数：
-t  ：后面可以接数字。一般来说，一个 tab 按键可以用 8 个空白键取代。
      我们也可以自行定义一个 [tab] 按键代表多少个字符呢！

范例一：将 /etc/man_db.conf 内行首为 MANPATH 的字样就取出；仅取前三行；
[dmtsai@study ~]$ grep '^MANPATH' /etc/man_db.conf &#124; head -n 3
MANPATH_MAP     /bin                    /usr/share/man
MANPATH_MAP     /usr/bin                /usr/share/man
MANPATH_MAP     /sbin                   /usr/share/man
# 行首的代表标志为 ^ ，这个我们留待下节介绍！先有概念即可！

范例二：承上，如果我想要将所有的符号都列出来？（用 cat）
[dmtsai@study ~]$ grep '^MANPATH' /etc/man_db.conf &#124; head -n 3 &#124;cat -A
MANPATH_MAP^I/bin^I^I^I/usr/share/man$
MANPATH_MAP^I/usr/bin^I^I/usr/share/man$
MANPATH_MAP^I/sbin^I^I^I/usr/share/man$
# 发现差别了吗？没错～ [tab] 按键可以被 cat -A 显示成为 ^I

范例三：承上，我将 [tab] 按键设置成 6 个字符的话？
[dmtsai@study ~]$ grep '^MANPATH' /etc/man_db.conf &#124; head -n 3 &#124; expand -t 6 -
MANPATH_MAP /bin              /usr/share/man$
MANPATH_MAP /usr/bin          /usr/share/man$
MANPATH_MAP /sbin             /usr/share/man$
123456123456123456123456123456123456123456123456...
# 仔细看一下上面的数字说明，因为我是以 6 个字符来代表一个 [tab] 的长度，所以，
# MAN... 到 /usr 之间会隔 12 （两个 [tab]） 个字符喔！如果 tab 改成 9 的话，
# 情况就又不同了！这里也不好理解～您可以多设置几个数字来查阅就晓得！
```

expand 也是挺好玩的～他会自动将 [tab] 转成空白键～所以，以上面的例子来说， 使用 cat -A 就会查不到 ^I 的字符啰～此外，因为 [tab] 最大的功能就是格式排列整齐！ 我们转成空白键后，这个空白键也会依据我们自己的定义来增加大小～ 所以，并不是一个 ^I 就会换成 8 个空白喔！这个地方要特别注意的哩！ 此外，您也可以参考一下 unexpand 这个将空白转成 [tab] 的指令功能啊！ ^_^

10.6.5 分区命令： split

如果你有文件太大，导致一些携带式设备无法复制的问题，嘿嘿！找 split 就对了！ 他可以帮你将一个大文件，依据文件大小或行数来分区，就可以将大文件分区成为小文件了！ 快速又有效啊！真不错～

```
[dmtsai@study ~]$ split [-bl] file PREFIX
选项与参数：
-b  ：后面可接欲分区成的文件大小，可加单位，例如 b, k, m 等；
-l  ：以行数来进行分区。
PREFIX ：代表前置字符的意思，可作为分区文件的前导文字。

范例一：我的 /etc/services 有六百多K，若想要分成 300K 一个文件时？
[dmtsai@study ~]$ cd /tmp; split -b 300k /etc/services services
[dmtsai@study tmp]$ ll -k services*
-rw-rw-r--. 1 dmtsai dmtsai 307200 Jul  9 22:52 servicesaa
-rw-rw-r--. 1 dmtsai dmtsai 307200 Jul  9 22:52 servicesab
-rw-rw-r--. 1 dmtsai dmtsai  55893 Jul  9 22:52 servicesac
# 那个文件名可以随意取的啦！我们只要写上前导文字，小文件就会以
# xxxaa, xxxab, xxxac 等方式来创建小文件的！

范例二：如何将上面的三个小文件合成一个文件，文件名为 servicesback
[dmtsai@study tmp]$ cat services* &gt;&gt; servicesback
# 很简单吧？就用数据流重导向就好啦！简单！

范例三：使用 ls -al / 输出的信息中，每十行记录成一个文件
[dmtsai@study tmp]$ ls -al / &#124; split -l 10 - lsroot
[dmtsai@study tmp]$ wc -l lsroot*
  10 lsrootaa
  10 lsrootab
   4 lsrootac
  24 total
# 重点在那个 - 啦！一般来说，如果需要 stdout/stdin 时，但偏偏又没有文件，
# 有的只是 - 时，那么那个 - 就会被当成 stdin 或 stdout ～
```

在 Windows 操作系统下，你要将文件分区需要如何作？伤脑筋吧！在 Linux 下面就简单的多了！你要将文件分区的话，那么就使用 -b size 来将一个分区的文件限制其大小，如果是行数的话，那么就使用 -l line 来分区！好用的很！如此一来，你就可以轻易的将你的文件分区成某些软件能够支持的最大容量 （例如 gmail 单一信件 25MB 之类的！），方便你 copy 啰！

10.6.6 参数代换： xargs

xargs 是在做什么的呢？就以字面上的意义来看， x 是加减乘除的乘号，args 则是 arguments （参数） 的意思，所以说，这个玩意儿就是在产生某个指令的参数的意思！ xargs 可以读入 stdin 的数据，并且以空白字符或断行字符作为分辨，将 stdin 的数据分隔成为 arguments 。 因为是以空白字符作为分隔，所以，如果有一些文件名或者是其他意义的名词内含有空白字符的时候， xargs 可能就会误判了～他的用法其实也还满简单的！就来看一看先！

```
[dmtsai@study ~]$ xargs [-0epn] command
选项与参数：
-0  ：如果输入的 stdin 含有特殊字符，例如 `, \, 空白键等等字符时，这个 -0 参数
      可以将他还原成一般字符。这个参数可以用于特殊状态喔！
-e  ：这个是 EOF （end of file） 的意思。后面可以接一个字串，当 xargs 分析到这个字串时，
      就会停止继续工作！
-p  ：在执行每个指令的 argument 时，都会询问使用者的意思；
-n  ：后面接次数，每次 command 指令执行时，要使用几个参数的意思。
当 xargs 后面没有接任何的指令时，默认是以 echo 来进行输出喔！

范例一：将 /etc/passwd 内的第一栏取出，仅取三行，使用 id 这个指令将每个帐号内容秀出来
[dmtsai@study ~]$ id root
uid=0（root） gid=0（root） groups=0（root）   # 这个 id 指令可以查询使用者的 UID/GID 等信息

[dmtsai@study ~]$ id $（cut -d ':' -f 1 /etc/passwd &#124; head -n 3）
# 虽然使用 $（cmd） 可以预先取得参数，但可惜的是， id 这个指令“仅”能接受一个参数而已！
# 所以上述的这个指令执行会出现错误！根本不会显示用户的 ID 啊！

[dmtsai@study ~]$ cut -d ':' -f 1 /etc/passwd &#124; head -n 3 &#124; id
uid=1000（dmtsai） gid=1000（dmtsai） groups=1000（dmtsai）,10（wheel）   # 我不是要查自己啊！
# 因为 id 并不是管线命令，因此在上面这个指令执行后，前面的东西通通不见！只会执行 id！

[dmtsai@study ~]$ cut -d ':' -f 1 /etc/passwd &#124; head -n 3 &#124; xargs id
# 依旧会出现错误！这是因为 xargs 一口气将全部的数据通通丢给 id 处理～但 id 就接受 1 个啊最多！

[dmtsai@study ~]$ cut -d ':' -f 1 /etc/passwd &#124; head -n 3 &#124; xargs -n 1 id
uid=0（root） gid=0（root） groups=0（root）
uid=1（bin） gid=1（bin） groups=1（bin）
uid=2（daemon） gid=2（daemon） groups=2（daemon）
# 通过 -n 来处理，一次给予一个参数，因此上述的结果就 OK 正常的显示啰！

范例二：同上，但是每次执行 id 时，都要询问使用者是否动作？
[dmtsai@study ~]$ cut -d ':' -f 1 /etc/passwd &#124; head -n 3 &#124; xargs -p -n 1 id
id root ?...y
uid=0（root） gid=0（root） groups=0（root）
id bin ?...y
.....（下面省略）.....
# 呵呵！这个 -p 的选项可以让使用者的使用过程中，被询问到每个指令是否执行！

范例三：将所有的 /etc/passwd 内的帐号都以 id 查阅，但查到 sync 就结束指令串
[dmtsai@study ~]$ cut -d ':' -f 1 /etc/passwd &#124; xargs -e'sync' -n 1 id
# 仔细与上面的案例做比较。也同时注意，那个 -e'sync' 是连在一起的，中间没有空白键。
# 上个例子当中，第六个参数是 sync 啊，那么我们下达 -e'sync' 后，则分析到 sync 这个字串时，
# 后面的其他 stdin 的内容就会被 xargs 舍弃掉了！
```

其实，在 man xargs 里面就有三四个小范例，您可以自行参考一下内容。 此外， xargs 真的是很好用的一个玩意儿！您真的需要好好的参详参详！会使用 xargs 的原因是， 很多指令其实并不支持管线命令，因此我们可以通过 xargs 来提供该指令引用 standard input 之用！举例来说，我们使用如下的范例来说明：

```
范例四：找出 /usr/sbin 下面具有特殊权限的文件名，并使用 ls -l 列出详细属性
[dmtsai@study ~]$ find /usr/sbin -perm /7000 &#124; xargs ls -l
-rwx--s--x. 1 root lock      11208 Jun 10  2014 /usr/sbin/lockdev
-rwsr-xr-x. 1 root root     113400 Mar  6 12:17 /usr/sbin/mount.nfs
-rwxr-sr-x. 1 root root      11208 Mar  6 11:05 /usr/sbin/netreport
.....（下面省略）.....
# 聪明的读者应该会想到使用“ ls -l $（find /usr/sbin -perm /7000） ”来处理这个范例！
# 都 OK！能解决问题的方法，就是好方法！
```

10.6.7 关于减号 - 的用途

管线命令在 bash 的连续的处理程序中是相当重要的！另外，在 log file 的分析当中也是相当重要的一环， 所以请特别留意！另外，在管线命令当中，常常会使用到前一个指令的 stdout 作为这次的 stdin ， 某些指令需要用到文件名称 （例如 tar） 来进行处理时，该 stdin 与 stdout 可以利用减号 "-" 来替代， 举例来说：

```
[root@study ~]# mkdir /tmp/homeback
[root@study ~]# tar -cvf - /home &#124; tar -xvf - -C /tmp/homeback
```

上面这个例子是说：“我将 /home 里面的文件给他打包，但打包的数据不是纪录到文件，而是传送到 stdout； 经过管线后，将 tar -cvf - /home 传送给后面的 tar -xvf - ”。后面的这个 - 则是取用前一个指令的 stdout， 因此，我们就不需要使用 filename 了！这是很常见的例子喔！注意注意！

10.7 重点回顾

- 由于核心在内存中是受保护的区块，因此我们必须要通过“ Shell ”将我们输入的指令与 Kernel 沟通，好让 Kernel 可以控制硬件来正确无误的工作
- 学习 shell 的原因主要有：命令行的 shell 在各大 distribution 都一样；远端管理时命令行速度较快； shell 是管理 Linux 系统非常重要的一环，因为 Linux 内很多控制都是以 shell 撰写的。
- 系统合法的 shell 均写在 /etc/shells 文件中；
- 使用者默认登陆取得的 shell 记录于 /etc/passwd 的最后一个字段；
- bash 的功能主要有：命令编修能力；命令与文件补全功能；命令别名设置功能；工作控制、前景背景控制；程序化脚本；万用字符
- type 可以用来找到执行指令为何种类型，亦可用于与 which 相同的功能；
- 变量就是以一组文字或符号等，来取代一些设置或者是一串保留的数据
- 变量主要有环境变量与自订变量，或称为全域变量与区域变量
- 使用 env 与 export 可观察环境变量，其中 export 可以将自订变量转成环境变量；
- set 可以观察目前 bash 环境下的所有变量；
- $? 亦为变量，是前一个指令执行完毕后的回传值。在 Linux 回传值为 0 代表执行成功；
- locale 可用于观察语系数据；
- 可用 read 让使用者由键盘输入变量的值
- ulimit 可用以限制使用者使用系统的资源情况
- bash 的配置文件主要分为 login shell 与 non-login shell。login shell 主要读取 /etc/profile 与 ~/.bash_profile， non-login shell 则仅读取 ~/.bashrc
- 在使用 vim 时，若不小心按了 [crtl]+s 则画面会被冻结。你可以使用 [ctrl]+q 来解除冻结
- 万用字符主要有： *, ?, [] 等等
- 数据流重导向通过 >, 2>, < 之类的符号将输出的信息转到其他文件或设备去；
- 连续命令的下达可通过 ; && || 等符号来处理
- 管线命令的重点是：“管线命令仅会处理 standard output，对于 standard error output 会予以忽略” “管线命令必须要能够接受来自前一个指令的数据成为 standard input 继续处理才行。”
- 本章介绍的管线命令主要有：cut, grep, sort, wc, uniq, tee, tr, col, join, paste, expand, split, xargs 等。

10.8 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ）

- 情境仿真题一：由于 ~/.bash_history 仅能记录指令，我想要在每次登出时都记录时间，并将后续的指令 50 笔记录下来， 可以如何处理？
    - 目标：了解 history ，并通过数据流重导向的方式记录历史命令；
    - 前提：需要了解本章的数据流重导向，以及了解 bash 的各个环境配置文件信息。

    其实处理的方式非常简单，我们可以了解 date 可以输出时间，而利用 ~/.myhistory 来记录所有历史记录， 而目前最新的 50 笔历史记录可以使用 history 50 来显示，故可以修改 ~/.bash_logout 成为下面的模样：

```
[dmtsai@study ~]$ vim ~/.bash_logout
date &gt;&gt; ~/.myhistory
history 50 &gt;&gt; ~/.myhistory
clear
```

简答题部分：

- 在 Linux 上可以找到哪些 shell（举出三个） ？那个文件记录可用的 shell ？而 Linux 默认的 shell 是？1） /bin/bash, /bin/tcsh, /bin/csh 2） /etc/shells 3） bash ，亦即是 /bin/bash。
- 你输入一串指令之后，发现前面写的一长串数据是错的，你想要删除光标所在处到最前面的指令串内容，应该如何处理？按下 [crtl]+u 组合键即可！
- 在 shell 环境下，有个提示字符 （prompt），他可以修改吗？要改什么？默认的提示字符内容是？可以修改的，改 PS1 这个变量，这个 PS1 变量的默认内容为：“[\u@\h \W]$”
- 如何显示 HOME 这个环境变量？echo $HOME
- 如何得知目前的所有变量与环境变量的设置值？环境变量用 env 或 export 而所有变量用 set 即可显示
- 我是否可以设置一个变量名称为 3myhome ？不行！变量不能以数字做为开头，参考变量设置规则的内容
- 在这样的练习中“A=B”且“B=C”，若我下达“unset $A”，则取消的变量是 A 还是 B？被取消的是 B 喔，因为 unset $A 相当于 unset B 所以取消的是 B ，A 会继续存在！
- 如何取消变量与命令别名的内容？使用 unset 及 unalias 即可
- 如何设置一个变量名称为 name 内容为 It's my name ？name=It\'s\ my\ name 或 name="It's my name"
- bash 环境配置文件主要分为哪两种类型的读取？分别读取哪些重要文件？（1）login shell：主要读取 /etc/profile 及 ~/.bash_profile （2）non-logni shell：主要读取 ~/.bashrc 而已。
- CentOS 7.x 的 man page 的路径设置文件？/etc/man_db.conf

- 试说明 ', ", 与 `这些符号在变量定义中的用途？参考变量规则那一章节，其中， " 可以具有变量的内容属性，' 则仅有一般字符，` 之内则是可先被执行的指令。
- 跳脱符号 \ 有什么用途？可以用来跳脱特殊字符，例如 Enter, $ 等等，使成为一般字符！
- 连续命令中， ;, &&, || 有何不同？分号可以让两个 command 连续运行，不考虑 command1 的输出状态， && 则前一个指令必需要没有错误讯息，亦即回传值需为 0 则 command2 才会被执行， || 则与 && 相反！
- 如何将 last 的结果中，独立出帐号，并且印出曾经登陆过的帐号？ last | cut -d ' ' -f1 | sort | uniq
- 请问 foo1 && foo2 | foo3 > foo4 ，这个指令串当中， foo1/foo2/foo3/foo4 是指令还是文件？ 整串指令的意义为？foo1, foo2 与 foo3 都是指令， foo4 是设备或文件。整串指令意义为：（1）当 foo1 执行结果有错误时，则该指令串结束；（2）若 foo1 执行结果没有错误时，则执行 foo2 | foo3 > foo4 ；其中：（2-1）foo2 将 stdout 输出的结果传给 foo3 处理；（2-2）foo3 将来自 foo2 的 stdout 当成 stdin ，处理完后将数据流重新导向 foo4 这个设备/文件
- 如何秀出在 /bin 下面任何以 a 为开头的文件文件名的详细数据？ls -ld /bin/a*
- 如何秀出 /bin 下面，文件名为四个字符的文件？ls -ld /bin/????
- 如何秀出 /bin 下面，文件名开头不是 a-d 的文件？ls -ld /bin/a-d*
- 我想要让终端机接口的登陆提示字符修改成我自己喜好的模样，应该要改哪里？（filename）/etc/issue
- 承上题，如果我是想要让使用者登陆后，才显示欢迎讯息，又应该要改哪里？/etc/motd

10.9 参考资料与延伸阅读

- [1]Webmin 的官方网站：http://www.webmin.com/
- [2]关于 shell 的相关历史可以参考网络农夫兄所整理的优秀文章。不过由于网络农夫兄所创建的网站暂时关闭， 因此下面的链接为鸟哥到网络上找到的片段文章链接。若有任何侵权事宜，请来信告知，谢谢： http://linux.vbird.org/linux_basic/0320bash/csh/
- [3]使用 man bash，再以 PS1 为关键字去查询，按下数次 n 往后查询后，可以得到 PS1 的变量说明。
- 在语系数据方面，i18n 是由一些 Linux distribution 贡献者共同发起的大型计划，目的在于让众多的 Linux distributions 能够有良好的万国码 （Unicode） 语系的支持。详细的数据可以参考：
    - i18n 的 wiki 介绍：https://en.wikipedia.org/wiki/Internationalization_and_localization
    - 康桥大学 Dr Markus Kuhn 的文献：http://www.cl.cam.ac.uk/~mgk25/unicode.html
    - Debian 社群所写的文件：http://www.debian.org/doc/manuals/intro-i18n/
- GNU 计划的 BASH 说明：http://www.gnu.org/software/bash/manual/bash.html
- man bash

2002/06/27：第一次完成 2003/02/10：重新编排与加入 FAQ 2005/08/17：将旧的数据放置到这里 2005/08/17：终于稍微搞定了～花了半个多月不眠不休～呼～补充了较多的管线命令与数据流重导向！ 2005/08/18：加入额外的变量设置部分！ 2005/08/30：加入了 login 与 non-login shell 的简单说明！ 2006/03/19：原先在 col 的说明当中，原本指令“cat -A /etc/man.config | col -x | cat -A | more”不该有 -A！ 2006/10/05：感谢小州兄的告知，修正了原本 ~/.bashrc 说明当中的错误。 2007/04/05：原本的 cut 范例说明有误，原本是“我要找出第三个”应该改为“我要找出第五个”才对！ 2007/04/11：原本的 join 说明没有加上排序，应该需要排序后再处理才对！ 2007/07/15：原本的额外的变量功能表格有误，在 var=${str+expr} 与var=${str:+expr} 需要修改，请参考 此处 2009/01/13：将原本基于 FC4 写作的旧文章移动到此处 2009/02/03：拿掉了原本的“变量的用途”部分，改以案例说明 2009/02/05：多加了变量删除、取代与替代部分的范例，看起来应该不会像前一版那样不容易理解！ 2009/08/25：加入了情境仿真，并且进行一些说明的细部修改而已。 2010/04/16：感谢wenyenyang兄的告知， wc -c 错误，是 wc -m 才是！

第十一章、正则表达式与文件格式化处理

最近更新日期：20//

正则表达式 （Regular Expression, RE, 或称为常规表达式）是通过一些特殊字符的排列，用以“搜寻/取代/删除”一列或多列文字字串， 简单的说，正则表达式就是用在字串的处理上面的一项“表示式”。正则表达式并不是一个工具程序， 而是一个字串处理的标准依据，如果您想要以正则表达式的方式处理字串，就得要使用支持正则表达式的工具程序才行， 这类的工具程序很多，例如 vi, sed, awk 等等。

正则表达式对于系统管理员来说实在是很重要！因为系统会产生很多的讯息，这些讯息有的重要有的仅是告知， 此时，管理员可以通过正则表达式的功能来将重要讯息撷取出来，并产生便于查阅的报表来简化管理流程。此外， 很多的套装软件也都支持正则表达式的分析，例如邮件服务器的过滤机制（过滤垃圾信件）就是很重要的一个例子。 所以，您最好要了解正则表达式的相关技能，在未来管理主机时，才能够更精简处理您的日常事务！

本章节使用者需要多加练习，因为目前很多的套件都是使用正则表达式来达成其“过滤、分析”的目的， 为了未来主机管理的便利性，使用者至少要能看的懂正则表达式的意义！

11.1 开始之前：什么是正则表达式

约略了解了 Linux 的基本指令（BASH）并且熟悉了 vim 之后，相信你对于敲击键盘的打字与指令下达比较不陌生了吧？接下来，下面要开始介绍一个很重要的观念，那就是所谓的“正则表达式（Regular Expression）”啰！

- 什么是正则表达式

任何一个有经验的系统管理员，都会告诉你：“正则表达式真是挺重要的！”为什么很重要呢？因为日常生活就使用的到啊！举个例子来说，在你日常使用 vim 作文书处理或程序撰写时使用到的“搜寻/取代”等等的功能，这些举动要作的漂亮，就得要配合正则表达式来处理啰！

简单的说，正则表达式就是处理字串的方法，他是以行为单位来进行字串的处理行为，正则表达式通过一些特殊符号的辅助，可以让使用者轻易的达到“搜寻/删除/取代”某特定字串的处理程序！

举例来说，我只想找到 VBird（前面两个大写字符）或 Vbird（仅有一个大写字符）这个字样，但是不要其他的字串（例如 VBIRD, vbird 等不需要），该如何办理？如果在没有正则表达式的环境中（例如 MS word），你或许就得要使用忽略大小写的办法，或者是分别以 VBird 及 Vbird 搜寻两遍。但是，忽略大小写可能会搜寻到 VBIRD/vbird/VbIrD 等等的不需要的字串而造成困扰。

再举个系统常见的例子好了，假设你发现系统在开机的时候，老是会出现一个关于 mail 程序的错误，而开机过程的相关程序都是在 /lib/systemd/system/ 下面，也就是说，在该目录下面的某个文件内具有 mail 这个关键字，你想要将该文件捉出来进行查询修改的动作。此时你怎么找出来含有这个关键字的文件？你当然可以一个文件一个文件的打开，然后去搜寻 mail 这个关键字，只是.....该目录下面的文件可能不止 100 个说～如果了解正则表达式的相关技巧，那么只要一行指令就找出来啦：“grep 'mail' /lib/systemd/system/*”那个 grep 就是支持正则表达式的工具程序之一！如何～很简单吧！

谈到这里就得要进一步说明了，正则表达式基本上是一种“表达式”，只要工具程序支持这种表达式，那么该工具程序就可以用来作为正则表达式的字串处理之用。例如 vi, grep, awk ,sed 等等工具，因为她们有支持正则表达式，所以，这些工具就可以使用正则表达式的特殊字符来进行字串的处理。但例如 cp, ls 等指令并未支持正则表达式，所以就只能使用 bash 自己本身的万用字符而已。

- 正则表达式对于系统管理员的用途

那么为何我需要学习正则表达式呢？对于一般使用者来说，由于使用到正则表达式的机会可能不怎么多，因此感受不到他的魅力，不过，对于身为系统管理员的你来说，正则表达式则是一个“不可不学的好东西！”怎么说呢？由于系统如果在繁忙的情况之下，每天产生的讯息

信息会多到你无法想像的地步， 而我们也都知道，系统的“错误讯息登录文件 （第十八章）”的内容记载了系统产生的所有讯息，当然，这包含你的系统是否被“入侵”的记录数据。

但是系统的数据量太大了，要身为系统管理员的你每天去看这么多的讯息数据， 从千百行的数据里面找出一行有问题的讯息，呵呵～光是用肉眼去看，想不疯掉都很难！ 这个时候，我们就可以通过“正则表达式”的功能，将这些登录的信息进行处理， 仅取出“有问题”的信息来进行分析，哈哈！如此一来，你的系统管理工作将会 “快乐得不得了”啊！当然，正则表达式的优点还不止于此，等你有一定程度的了解之后，你会爱上他喔！

- 正则表达式的广泛用途

正则表达式除了可以让系统管理员管理主机更为便利之外，事实上，由于正则表达式强大的字串处理能力， 目前一堆软件都支持正则表达式呢！最常见的就是“邮件服务器”啦！

如果你留意网际网络上的消息，那么应该不难发现，目前造成网络大塞车的主因之一就是“垃圾/广告信件”了， 而如果我们可以在服务器端，就将这些问题邮件剔除的话，用户端就会减少很多不必要的带宽耗损了。 那么如何剔除广告信件呢？由于广告信件几乎都有一定的标题或者是内容，因此， 只要每次有来信时，都先将来信的标题与内容进行特殊字串的比对，发现有不良信件就予以剔除！ 嘿！这个工作怎么达到啊？就使用正则表达式啊！目前两大邮件服务器软件 sendmail 与 postfix 以及支持邮件服务器的相关分析软件，都支持正则表达式的比对功能！

当然还不止于此啦，很多的服务器软件都支持正则表达式呢！当然， 虽然各家软件都支持他，不过，这些“字串”的比对还是需要系统管理员来加入比对规则的， 所以啦！身为系统管理员的你，为了自身的工作以及用户端的需求， 正则表达式实在是很需要也很值得学习的一项工具呢！

- 正则表达式与 Shell 在 Linux 当中的角色定位

说实在的，我们在学数学的时候，一个很重要、但是粉难的东西是一定要“背”的， 那就是九九乘法表，背成功了之后，未来在数学应用的路途上，真是一帆风顺啊！ 这个九九乘法表我们在小学的时候几乎背了一整年才背下来，并不是这么好背的呢！ 但他却是基础当中的基础！你现在一定受惠相当的多呢 ^_^！

而我们谈到的这个正则表达式，与前一章的 BASH 就有点像是数学的九九乘法表一样，是 Linux 基础当中的基础，虽然也是最难的部分， 不过，如果学成了之后，一定是“大大的有帮助”的！这就好像是金庸小说里面的学武难关：任督二脉！ 打通任督二脉之后，武功立刻成倍成长！所以啦， 不论是对于系统的认识与系统的管理部分，他都有很棒的辅助啊！请好好的学习这个基础吧！ ^_^

- 延伸的正则表达式

唔！正则表达式还有分喔？没错喔！正则表达式的字串表示方式依照不同的严谨度而分为：基础正则表达式与延伸正则表达式。延伸型正则表达式除了简单的一组字串处理之外，还可以作群组的字串处理， 例如进行搜寻 VBird 或 netman 或 lman 的搜寻，注意，是“或

（or）”而不是“和（and）”的处理， 此时就需要延伸正则表达式的帮助啦！借由特殊的“（”与“|”等字符的协助， 就能够达到这样的目的！不过，我们在这里主力仅是介绍最基础的基础正则表达式而已啦！好啦！清清脑门，咱们用功去啰！



Tips 有一点要向大家报告的，那就是：“正则表达式与万用字符是完全不一样的东西！” 这很重要喔！因为“万用字符 （wildcard） 代表的是 bash 操作接口的一个功能”，但正则表达式则是一种字串处理的表示方式！ 这两者要分的很清楚才行喔！所以，学习本章，请将前一章 bash 的万用字符意义先忘掉吧！

老实说，鸟哥以前刚接触正则表达式时，老想着要将这两者归纳在一起，结果就是...错误认知一大堆～ 所以才会建议您学习本章先忘记万用字符再来学习吧！

11.2 基础正则表达式

既然正则表达式是处理字串的一种表示方式，那么对字符排序有影响的语系数据就会对正则表达式的结果有影响！ 此外，正则表达式也需要支持工具程序来辅助才行！所以，我们这里就先介绍一个最简单的字串撷取功能的工具程序，那就是 grep 啰！ 前一章已经介绍过 grep 的相关选项与参数，本章着重在较进阶的 grep 选项说明啰！ 介绍完 grep 的功能之后，就进入正则表达式的特殊字符的处理能力了。

11.2.1 语系对正则表达式的影响

为什么语系的数据会影响到正则表达式的输出结果呢？我们在第零章计算机概论的文字编码系统里面谈到，文件其实记录的仅有 0 与 1，我们看到的字符文字与数字都是通过编码表转换来的。由于不同语系的编码数据并不相同，所以就会造成数据撷取结果的差异了。 举例来说，在英文大小写的编码顺序中，zh_TW.big5 及 C 这两种语系的输出结果分别如下：

- LANG=C 时：0 1 2 3 4 ... A B C D ... Z a b c d ...z
- LANG=zh_TW 时：0 1 2 3 4 ... a A b B c C d D ... z Z

上面的顺序是编码的顺序，我们可以很清楚的发现这两种语系明显就是不一样！如果你想要撷取大写字符而使用 [A-Z] 时， 会发现 LANG=C 确实可以仅捉到大写字符 （因为是连续的），但是如果 LANG=zh_TW.big5 时，就会发现到， 连同小写的 b-z 也会被撷取出来！因为就编码的顺序来看， big5 语系可以撷取到“ A b B c C ... z Z ”这一堆字符哩！ 所以，使用正则表达式时，需要特别留意当时环境的语系为何， 否则可能会发现与别人不相同的撷取结果喔！

由于一般我们在练习正则表达式时，使用的是相容于 POSIX 的标准，因此就使用“ C ”这个语系[1]！ 因此，下面的很多练习都是使用“ LANG=C ”这个语系数据来进行的喔！ 另外，为了要避免这样编码所造成的英文与数字的撷取问题，因此有些特殊的符号我们得要了解一下的！ 这些符号主要有下面这些意义：

特殊符号	代表意义
[:alnum:]	代表英文大小写字符及数字，亦即 0-9, A-Z, a-z
[:alpha:]	代表任何英文大小写字符，亦即 A-Z, a-z
[:blank:]	代表空白键与 [Tab] 按键两者
[:cntrl:]	代表键盘上面的控制按键，亦即包括 CR, LF, Tab, Del.. 等等
[:digit:]	代表数字而已，亦即 0-9
[:graph:]	除了空白字符 （空白键与 [Tab] 按键） 外的其他所有按键
[:lower:]	代表小写字符，亦即 a-z
[:print:]	代表任何可以被打印出来的字符
[:punct:]	代表标点符号 （punctuation symbol），亦即：" ' ? ! ; : # $...
[:upper:]	代表大写字符，亦即 A-Z
[:space:]	任何会产生空白的字符，包括空白键, [Tab], CR 等等
[:xdigit:]	代表 16 进位的数字类型，因此包括： 0-9, A-F, a-f 的数字与字符

尤其上表中的[:alnum:], [:alpha:], [:upper:], [:lower:], [:digit:] 这几个一定要知道代表什么意思，因为他要比 a-z 或 A-Z 的用途要确定的很！好了，下面就让我们开始来玩玩进阶版的 grep 吧！

11.2.2 grep 的一些进阶选项

我们在第十章 BASH 里面的 grep 谈论过一些基础用法， 但其实 grep 还有不少的进阶用法喔！下面我们仅列出较进阶的 grep 选项与参数给大家参考， 基础的 grep 用法请参考前一章的说明啰！

```
[dmtsai@study ~]$ grep [-A] [-B] [--color=auto] '搜寻字串' filename
选项与参数：
-A ：后面可加数字，为 after 的意思，除了列出该行外，后续的 n 行也列出来；
-B ：后面可加数字，为 befer 的意思，除了列出该行外，前面的 n 行也列出来；
--color=auto 可将正确的那个撷取数据列出颜色

范例一：用 dmesg 列出核心讯息，再以 grep 找出内含 qxl 那行
[dmtsai@study ~]$ dmesg &#124; grep 'qxl'
[    0.522749] [drm] qxl: 16M of VRAM memory size
[    0.522750] [drm] qxl: 63M of IO pages memory ready （VRAM domain）
[    0.522750] [drm] qxl: 32M of Surface memory size
[    0.650714] fbcon: qxldrmfb （fb0） is primary device
[    0.668487] qxl 0000:00:02.0: fb0: qxldrmfb frame buffer device
# dmesg 可列出核心产生的讯息！包括硬件侦测的流程也会显示出来。
# 鸟哥使用的显卡是 QXL 这个虚拟卡，通过 grep 来 qxl 的相关信息，可发现如上信息。

范例二：承上题，要将捉到的关键字显色，且加上行号来表示：
[dmtsai@study ~]$ dmesg &#124; grep -n --color=auto 'qxl'
515:[    0.522749] [drm] qxl: 16M of VRAM memory size
516:[    0.522750] [drm] qxl: 63M of IO pages memory ready （VRAM domain）
517:[    0.522750] [drm] qxl: 32M of Surface memory size
529:[    0.650714] fbcon: qxldrmfb （fb0） is primary device
539:[    0.668487] qxl 0000:00:02.0: fb0: qxldrmfb frame buffer device
# 除了 qxl 会有特殊颜色来表示之外，最前面还有行号喔！其实颜色显示已经是默认在 alias 当中了！

范例三：承上题，在关键字所在行的前两行与后三行也一起捉出来显示
[dmtsai@study ~]$ dmesg &#124; grep -n -A3 -B2 --color=auto 'qxl'
# 你会发现关键字之前与之后的数行也被显示出来！这样可以让你将关键字前后数据捉出来进行分析啦！
```

grep 是一个很常见也很常用的指令，他最重要的功能就是进行字串数据的比对，然后将符合使用者需求的字串行印出来。 需要说明的是“grep 在数据中查寻一个字串时，是以 "整行" 为单位来进行数据的撷取的！”也就是说，假如一个文件内有 10 行，其中有两行具有你所搜寻的字串，则将那两行显示在屏幕上，其他的就丢弃了！

在 CentOS 7 当中，默认已经将 --color=auto 加入在 alias 当中了！使用者就可以直接使用有关键字显色的 grep 啰！非常方便！

11.2.3 基础正则表达式练习

要了解正则表达式最简单的方法就是由实际练习去感受啦！所以在汇整正则表达式特殊符号前， 我们先以下面这个文件的内容来进行正则表达式的理解吧！先说明一下，下面的练习大前提是：

- 语系已经使用“ export LANG=C; export LC_ALL=C ”的设置值；
- grep 已经使用 alias 设置成为“ grep --color=auto ”

至于本章的练习用文件请由下面的链接来下载。需要特别注意的是，下面这个文件是鸟哥在 MS Windows 系统下编辑的， 并且已经特殊处理过，因此，他虽然是纯文本文件，但是内含一些 Windows 系统下的软件常常自行加入的一些特殊字符，例如断行字符 （^M） 就是一例！ 所以，你可以直接将下面的文字以 vi 储存成 regular_express.txt 这个文件， 不过，还是比较建议直接点下面的链接：

> http://linux.vbird.org/linux_basic/0330regularex/regular_express.txt

如果你的 Linux 可以直接连上 Internet 的话，那么使用如下的指令来捉取即可：

> wget http://linux.vbird.org/linux_basic/0330regularex/regular_express.txt

至于这个文件的内容如下：

```
[dmtsai@study ~]$ vi regular_express.txt
"Open Source" is a good mechanism to develop programs.
apple is my favorite food.
Football game is not use feet only.
this dress doesn't fit me.
However, this dress is about $ 3183 dollars.^M
GNU is free air not free beer.^M
Her hair is very beauty.^M
I can't finish the test.^M
Oh! The soup taste good.^M
motorcycle is cheap than car.
This window is clear.
the symbol '*' is represented as start.
Oh!     My god!
The gd software is a library for drafting programs.^M
You are the best is mean you are the no. 1.
The world &lt;Happy&gt; is the same with "glad".
I like dog.
google is the best tools for search keyword.
goooooogle yes!
go! go! Let's go.
# I am VBird
```

这文件共有 22 行，最下面一行为空白行！现在开始我们一个案例一个案例的来介绍吧！

- 例题一、搜寻特定字串

搜寻特定字串很简单吧？假设我们要从刚刚的文件当中取得 the 这个特定字串，最简单的方式就是这样：

```
[dmtsai@study ~]$ grep -n 'the' regular_express.txt
8:I can't finish the test.
12:the symbol '*' is represented as start.
15:You are the best is mean you are the no. 1.
16:The world &lt;Happy&gt; is the same with "glad".
18:google is the best tools for search keyword.
```

那如果想要“反向选择”呢？也就是说，当该行没有 'the' 这个字串时才显示在屏幕上，那就直接使用：

```
[dmtsai@study ~]$ grep -vn 'the' regular_express.txt
```

你会发现，屏幕上出现的行列为除了 8,12,15,16,18 五行之外的其他行列！ 接下来，如果你想要取得不论大小写的 the 这个字串，则：

```
[dmtsai@study ~]$ grep -in 'the' regular_express.txt
8:I can't finish the test.
9:Oh! The soup taste good.
12:the symbol '*' is represented as start.
14:The gd software is a library for drafting programs.
15:You are the best is mean you are the no. 1.
16:The world &lt;Happy&gt; is the same with "glad".
18:google is the best tools for search keyword.
```

除了多两行（9, 14行）之外，第 16 行也多了一个 The 的关键字被撷取到喔！

- 例题二、利用中括号 [] 来搜寻集合字符

如果我想要搜寻 test 或 taste 这两个单字时，可以发现到，其实她们有共通的 't?st' 存在～这个时候，我可以这样来搜寻：

```
[dmtsai@study ~]$ grep -n 't[ae]st' regular_express.txt
8:I can't finish the test.
9:Oh! The soup taste good.
```

了解了吧？其实 [] 里面不论有几个字符，他都仅代表某“一个”字符， 所以，上面的例子说明了，我需要的字串是“tast”或“test”两个字串而已！ 而如果想要搜寻到有 oo 的字符时，则使用：

```
[dmtsai@study ~]$ grep -n 'oo' regular_express.txt
1:"Open Source" is a good mechanism to develop programs.
2:apple is my favorite food.
3:Football game is not use feet only.
9:Oh! The soup taste good.
18:google is the best tools for search keyword.
19:goooooogle yes!
```

但是，如果我不想要 oo 前面有 g 的话呢？此时，可以利用在集合字符的反向选择 来达成：

```
[dmtsai@study ~]$ grep -n '[^g]oo' regular_express.txt
2:apple is my favorite food.
3:Football game is not use feet only.
18:google is the best tools for search keyword.
19:goooooogle yes!
```

意思就是说，我需要的是 oo ，但是 oo 前面不能是 g 就是了！仔细比较上面两个表格，你会发现，第 1,9 行不见了，因为 oo 前面出现了 g 所致！第 2,3 行没有疑问，因为 foo 与 Foo 均可被接受！但是第 18 行明明有 google 的 goo 啊～别忘记了，因为该行后面出现了 tool 的 too 啊！所以该行也被列出来～ 也就是说， 18 行里面虽然出现了我们所不要的项目 （goo） 但是由于有需要的项目 （too） ， 因此，是符合字串搜寻的喔！

至于第 19 行，同样的，因为 goooooogle 里面的 oo 前面可能是 o ，例如： go（ooo）oogle ，所以，这一行也是符合需求的！

再来，假设我 oo 前面不想要有小写字符，所以，我可以这样写 abcd....zoo ， 但是这样似乎不怎么方便，由于小写字符的 ASCII 上编码的顺序是连续的， 因此，我们可以将之简化为下面这样：

```
[dmtsai@study ~]$ grep -n '[^a-z]oo' regular_express.txt
3:Football game is not use feet only.
```

也就是说，当我们在一组集合字符中，如果该字符组是连续的，例如大写英文/小写英文/数字等等， 就可以使用[a-z],[A-Z],[0-9]等方式来书写，那么如果我们的要求字串是数字与英文呢？ 呵呵！就将他全部写在一起，变成：[a-zA-Z0-9]。例如，我们要取得有数字的那一行，就这样：

```
[dmtsai@study ~]$ grep -n '[0-9]' regular_express.txt
5:However, this dress is about $ 3183 dollars.
15:You are the best is mean you are the no. 1.
```

但由于考虑到语系对于编码顺序的影响，因此除了连续编码使用减号“ - ”之外， 你也可以使用如下的方法来取得前面两个测试的结果：

```
[dmtsai@study ~]$ grep -n '[^[:lower:]]oo' regular_express.txt
# 那个 [:lower:] 代表的就是 a-z 的意思！请参考前两小节的说明表格

[dmtsai@study ~]$ grep -n '[[:digit:]]' regular_express.txt
```

啥？上头在写啥东西呢？不要害怕！分开来瞧一瞧。我们知道 [:lower:] 就是 a-z 的意思，那么 [a-z] 当然就是 [[:lower:]] 啰！鸟哥第一次接触正则表达式的时候，看到两层中括号差点昏倒～完全看不懂！现在，请注意那个叠代的意义， 自然就能够比较清楚了解啰！

这样对于 [] 以及 以及 [] 当中的 - ，还有关于前面表格提到的特殊关键字有了解了吗？^_^！

- 例题三、行首与行尾字符 ^ $

我们在例题一当中，可以查询到一行字串里面有 the 的，那如果我想要让 the 只在行首列出呢？ 这个时候就得要使用定位字符了！我们可以这样做：

```
[dmtsai@study ~]$ grep -n '^the' regular_express.txt
12:the symbol '*' is represented as start.
```

此时，就只剩下第 12 行，因为只有第 12 行的行首是 the 开头啊～此外， 如果我想要开头是小写字符的那一行就列出呢？可以这样：

```
[dmtsai@study ~]$ grep -n '^[a-z]' regular_express.txt
2:apple is my favorite food.
4:this dress doesn't fit me.
10:motorcycle is cheap than car.
12:the symbol '*' is represented as start.
18:google is the best tools for search keyword.
19:goooooogle yes!
20:go! go! Let's go.
```

你可以发现我们可以捉到第一个字符都不是大写的！上面的指令也可以用如下的方式来取代的：

```
[dmtsai@study ~]$ grep -n '^[[:lower:]]' regular_express.txt
```

好！那如果我不想要开头是英文字母，则可以是这样：

```
[dmtsai@study ~]$ grep -n '^[^a-zA-Z]' regular_express.txt
1:"Open Source" is a good mechanism to develop programs.
21:# I am VBird
# 指令也可以是： grep -n '^[^[:alpha:]]' regular_express.txt
```

注意到了吧？那个 ^ 符号，在字符集合符号（括号[]）之内与之外是不同的！ 在 [] 内代表“反向选择”，在 [] 之外则代表定位在行首的意义！要分清楚喔！ 反过来思考，那如果我想要找出来，行尾结束为小数点 （.） 的那一行，该如何处理：

```
[dmtsai@study ~]$ grep -n '\.$' regular_express.txt
1:"Open Source" is a good mechanism to develop programs.
2:apple is my favorite food.
3:Football game is not use feet only.
4:this dress doesn't fit me.
10:motorcycle is cheap than car.
11:This window is clear.
12:the symbol '*' is represented as start.
15:You are the best is mean you are the no. 1.
16:The world &lt;Happy&gt; is the same with "glad".
17:I like dog.
18:google is the best tools for search keyword.
20:go! go! Let's go.
```

特别注意到，因为小数点具有其他意义（下面会介绍），所以必须要使用跳脱字符（\）来加以解除其特殊意义！ 不过，你或许会觉得奇怪，但是第 5~9 行最后面也是 . 啊～怎么无法打印出来？ 这里就牵涉到 Windows 平台的软件对于断行字符的判断问题了！我们使用 cat -A 将第五行拿出来看， 你会发现：

```
[dmtsai@study ~]$ cat -An regular_express.txt &#124; head -n 10 &#124; tail -n 6
     5  However, this dress is about $ 3183 dollars.^M$
     6  GNU is free air not free beer.^M$
     7  Her hair is very beauty.^M$
     8  I can't finish the test.^M$
     9  Oh! The soup taste good.^M$
    10  motorcycle is cheap than car.$
```

我们在第九章内谈到过断行字符在 Linux 与 Windows 上的差异， 在上面的表格中我们可以发现 5~9 行为 Windows 的断行字符 （^M$） ，而正常的 Linux 应该仅有第 10 行显示的那样 （$） 。所以啰，那个 . 自然就不是紧接在 $ 之前喔！也就捉不到 5~9 行了！这样可以了解 ^ 与 $ 的意义吗？ 好了，先不要看下面的解答，自己想一想，那么如果我想要找出来，哪一行是“空白行”， 也就是说，该行并没有输入任何数据，该如何搜寻？

```
[dmtsai@study ~]$ grep -n '^$' regular_express.txt
22:
```

因为只有行首跟行尾 （^$） ，所以，这样就可以找出空白行啦！再来，假设你已经知道在一个程序脚本 （shell script） 或者是配置文件当中，空白行与开头为 # 的那一行是注解，因此如果你要将数据列出给别人参考时， 可以将这些数据省略掉以节省保贵的纸张，那么你可以怎么作呢？ 我们以 /etc/rsyslog.conf 这个文件来作范例，你可以自行参考一下输出的结果：

```
[dmtsai@study ~]$ cat -n /etc/rsyslog.conf
# 在 CentOS 7 中，结果可以发现有 91 行的输出，很多空白行与 # 开头的注解行

[dmtsai@study ~]$ grep -v '^$' /etc/rsyslog.conf &#124; grep -v '^#'
# 结果仅有 14 行，其中第一个“ -v '^$' ”代表“不要空白行”，
# 第二个“ -v '^#' ”代表“不要开头是 # 的那行”喔！
```

是否节省很多版面啊？另外，你可能也会问，那为何不要出现 # 的符号的那行就直接舍弃呢？没办法！因为某些注解是与设置写在同一行的后面， 如果你只是抓 # 就予以去除，那就会将某些设置也同时移除了！那错误就大了～

- 例题四、任意一个字符 . 与重复字符 *

在第十章 bash 当中，我们知道万用字符 * 可以用来代表任意（0或多个）字符， 但是正则表达式并不是万用字符，两者之间是不相同的！ 至于正则表达式当中的“ . ”则代表“绝对有一个任意字符”的意思！这两个符号在正则表达式的意义如下：

- . （小数点）：代表“一定有一个任意字符”的意思；
-
    - （星星号）：代表“重复前一个字符， 0 到无穷多次”的意思，为组合形态

这样讲不好懂，我们直接做个练习吧！假设我需要找出 g??d 的字串，亦即共有四个字符， 起头是 g 而结束是 d ，我可以这样做：

```
[dmtsai@study ~]$ grep -n 'g..d' regular_express.txt
1:"Open Source" is a good mechanism to develop programs.
9:Oh! The soup taste good.
16:The world &lt;Happy&gt; is the same with "glad".
```

因为强调 g 与 d 之间一定要存在两个字符，因此，第 13 行的 god 与第 14 行的 gd 就不会被列出来啦！再来，如果我想要列出有 oo, ooo, oooo 等等的数据， 也就是说，至少要有两个（含） o 以上，该如何是好？是 o 还是 *oo* 还是 ooo* 呢？ 虽然你可以试看看结果， 不过结

果太占版面了 @_@ ，所以，我这里就直接说明。

因为 代表的是“重复 *0* 个或多个前面的 *RE* 字符”的意义， 因此，“*o*”代表的是：“拥有空字符或一个 o 以上的字符”， 特别注意，因为允许空字符（就是有没有字符都可以的意思），因此，“ grep -n 'o*' regular_express.txt ”将会把所有的数据都打印出来屏幕上！

那如果是“oo*”呢？则第一个 o 肯定必须要存在，第二个 o 则是可有可无的多个 o ， 所以，凡是含有 o, oo, ooo, oooo 等等，都可以被列出来～

同理，当我们需要“至少两个 o 以上的字串”时，就需要 ooo* ，亦即是：

```
[dmtsai@study ~]$ grep -n 'ooo*' regular_express.txt
1:"Open Source" is a good mechanism to develop programs.
2:apple is my favorite food.
3:Football game is not use feet only.
9:Oh! The soup taste good.
18:google is the best tools for search keyword.
19:goooooogle yes!
```

这样理解 * 的意义了吗？好了，现在出个练习，如果我想要字串开头与结尾都是 g，但是两个 g 之间仅能存在至少一个 o ，亦即是 gog, goog, gooog.... 等等，那该如何？

```
[dmtsai@study ~]$ grep -n 'goo*g' regular_express.txt
18:google is the best tools for search keyword.
19:goooooogle yes!
```

如此了解了吗？再来一题，如果我想要找出 g 开头与 g 结尾的字串，当中的字符可有可无，那该如何是好？是“g*g”吗？

```
[dmtsai@study ~]$ grep -n 'g*g' regular_express.txt
1:"Open Source" is a good mechanism to develop programs.
3:Football game is not use feet only.
9:Oh! The soup taste good.
13:Oh!  My god!
14:The gd software is a library for drafting programs.
16:The world &lt;Happy&gt; is the same with "glad".
17:I like dog.
18:google is the best tools for search keyword.
19:goooooogle yes!
20:go! go! Let's go.
```

但测试的结果竟然出现这么多行？太诡异了吧？其实一点也不诡异，因为 *gg* 里面的 *g* 代表“空字符或一个以上的 g” 在加上后面的 g ，因此，整个 RE 的内容就是 g, gg, ggg, gggg ，因此，只要该行当中拥有一个以上的 g 就符合所需了！

那该如何得到我们的 g....g 的需求呢？呵呵！就利用任意一个字符“.”啊！ 亦即是：“g.*g*”的作法，因为 可以是 0 或多个重复前面的字符，而 . 是任意字符，所以： “.* 就代表零个或多个任意字符”的意思啦！

```
[dmtsai@study ~]$ grep -n 'g.*g' regular_express.txt
1:"Open Source" is a good mechanism to develop programs.
14:The gd software is a library for drafting programs.
18:google is the best tools for search keyword.
19:goooooogle yes!
20:go! go! Let's go.
```

因为是代表 g 开头与 g 结尾，中间任意字符均可接受，所以，第 1, 14, 20 行是可接受的喔！这个 .* 的 RE 表示任意字符是很常见的，希望大家能够理解并且熟悉！ 再出一题，如果我想要找出“任意数字”的行列呢？因为仅有数字，所以就成为：

```
[dmtsai@study ~]$ grep -n '[0-9][0-9]*' regular_express.txt
5:However, this dress is about $ 3183 dollars.
15:You are the best is mean you are the no. 1.
```

虽然使用 grep -n '[0-9]' regular_express.txt 也可以得到相同的结果， 但鸟哥希望大家能够理解上面指令当中 RE 表达式的意义才好！

- 例题五、限定连续 RE 字符范围 {}

在上个例题当中，我们可以利用 . 与 RE 字符及 * 来设置 0 个到无限多个重复字符， 那如果我想要限制一个范围区间内的重复字符数呢？举例来说，我想要找出两个到五个 o 的连续字串，该如何作？这时候就得要使用到限定范围的字符 {} 了。 但因为 { 与 } 的符号在 shell 是有特殊意义的，因此， 我们必须要使用跳脱字符 \ 来让他失去特殊意义才行。 至于 {} 的语法是这样的，假设我要找到两个 o 的字串，可以是：

```
[dmtsai@study ~]$ grep -n 'o\{2\}' regular_express.txt
1:"Open Source" is a good mechanism to develop programs.
2:apple is my favorite food.
3:Football game is not use feet only.
9:Oh! The soup taste good.
18:google is the best tools for search keyword.
19:goooooogle yes!
```

这样看似乎与 ooo* 的字符没有什么差异啊？因为第 19 行有多个 o 依旧也出现了！ 好，那么换个搜寻的字串，假设我们要找出 g 后面接 2 到 5 个 o ，然后再接一个 g 的字串，他会是这样：

```
[dmtsai@study ~]$ grep -n 'go\{2,5\}g' regular_express.txt
18:google is the best tools for search keyword.
```

嗯！很好！第 19 行终于没有被取用了（因为 19 行有 6 个 o 啊！）。 那么，如果我想要的是 2 个 o 以上的 goooo....g 呢？除了可以是 gooo*g ，也可以是：

```
[dmtsai@study ~]$ grep -n 'go\{2,\}g' regular_express.txt
18:google is the best tools for search keyword.
19:goooooogle yes!
```

呵呵！就可以找出来啦～

11.2.4 基础正则表达式字符汇整 （characters）

经过了上面的几个简单的范例，我们可以将基础的正则表达式特殊字符汇整如下：

RE 字符	意义与范例
^word	意义：待搜寻的字串（word）在行首！范例：搜寻行首为 # 开始的那一行，并列出行号 `> grep -n '^#' regular_express.txt`
word$	意义：待搜寻的字串（word）在行尾！范例：将行尾为 ! 的那一行打印出来，并列出行号 `> grep -n '!$' regular_express.txt`
.	意义：代表"一定有一个任意字符"的字符！范例：搜寻的字串可以是 （eve）（eae）（eee）（e e），但不能仅有 （ee）！亦即 e 与 e 中间"一定"仅有一个字符，而空白字符也是字符！ `> grep -n 'e.e' regular_express.txt`
\	意义：跳脱字符，将特殊符号的特殊意义去除！范例：搜寻含有单引号 ' 的那一行！ `> grep -n \' regular_express.txt`
*	意义：重复零个到无穷多个的前一个 RE 字符 范例：找出含有 （es） （ess）（esss） 等等的字串，注意，因为 可以是 *0* 个，所以 *es* 也是符合带搜寻字串。另外，因为 为重复"前一个 RE 字符"的符号， 因此，在 之前必须要紧接着一个 *RE* 字符喔！例如任意字符则为 "*.*"！ `> grep -n 'ess*' regular_express.txt`
[list]	意义：字符集合的 RE 字符，里面列出想要撷取的字符！范例：搜寻含有 （gl） 或 （gd） 的那一行，需要特别留意的是，在 [] 当中"谨代表一个待搜寻的字符"， 例如" a[afl]y "代表搜寻的字串可以是 aay, afy, aly 即 [afl] 代表 a 或 f 或 l 的意思！ `> grep -n 'g[ld]' regular_express.txt`
[n1-n2]	意义：字符集合的 RE 字符，里面列出想要撷取的字符范围！范例：搜寻含有任意数字的那一行！需特别留意，在字符集合 [] 中的减号 - 是有特殊意义的，他代表两个字符之间的所有连续字符！但这个连续与否与 ASCII 编码有关，因此，你的编码需要设置正确（在 bash 当中，需要确定 LANG 与 LANGUAGE 的变量是否正确！） 例如所有大写字符则为 [A-Z] `> grep -n '[A-Z]' regular_express.txt`
[^list]	意义：字符集合的 RE 字符，里面列出不要的字串或范围！范例：搜寻的字串可以是 （oog） （ood） 但不能是 （oot） ，那个 ^ 在 [] 内时，代表的意义是"反向选择"的意思。 例如，我不要大写字符，则为 [^A-Z]。但是，需要特别注意的是，如果以 grep -n [^A-Z] regular_express.txt 来搜寻，却发现该文件内的所有行都被列出，为什么？因为这个 [^A-Z] 是"非大写字符"的意思， 因为每一行均有非大写字符，例如第一行的 "Open Source" 就有 p,e,n,o.... 等等的小写字 `> grep -n 'oo[^t]' regular_express.txt`
{n,m}	意义：连续 n 到 m 个的"前一个 RE 字符" 意义：若为 {n} 则是连续 n 个的前一个 RE 字符， 意义：若是 {n,} 则是连续 n 个以上的前一个 RE 字符！ 范例：在 g 与 g 之间有 2 个到 3 个的 o 存在的字串，亦即 （goog）（gooog） `> grep -n 'go\{2,3\}g' regular_express.txt`

再次强调："正则表达式的特殊字符"与一般在命令行输入指令的"万用字符"并不相同，例如，在万用字符当中的 代表的是" *0 ~* 无限多个字符"的意思，但是在正则表达式当中， 则是"重复 0 到无穷多个的前一个 RE 字符"的意思～使用的意义并不相同，不要搞混了！

举例来说，不支持正则表达式的 ls 这个工具中，若我们使用 "ls -l " 代表的是任意文件名的文件，而 *"ls -l a "*代表的是以 a 为开头的任何文件名的文件， 但在正则表达式中，我们要找到含有以 a 为开头的文件，则必须要这样：（需搭配支持正则表达式的工具）

> ls | grep -n '^a.*'

例题：以 ls -l 配合 grep 找出 /etc/ 下面文件类型为链接文件属性的文件名答：由于 ls -l 列出链接文件时标头会是" lrwxrwxrwx "，因此使用如下的指令即可找出结果：

> ls -l /etc | grep '^l'

若仅想要列出几个文件，再以" |wc -l " 来累加处理即可。

11.2.5 sed 工具

在了解了一些正则表达式的基础应用之后，再来呢？呵呵～两个东西可以玩一玩的，那就是 sed 跟下面会介绍的 awk 了！ 这两个家伙可是相当的有用的啊！举例来说，鸟哥写的 logfile.sh 分析登录文件的小程序 （第十八章会谈到），绝大部分分析关键字的取用、统计等等，就是用这两个宝贝蛋来帮我完成的！那么你说，要不要玩一玩啊？^_^

我们先来谈一谈 sed 好了， sed 本身也是一个管线命令，可以分析 standard input 的啦！ 而且 sed 还可以将数据进行取代、删除、新增、撷取特定行等等的功能呢！很不错吧～ 我们先来了解一下 sed 的用法，再来聊他的用途好了！

```
[dmtsai@study ~]$ sed [-nefr] [动作]
选项与参数：
-n  ：使用安静（silent）模式。在一般 sed 的用法中，所有来自 STDIN 的数据一般都会被列出到屏幕上。
      但如果加上 -n 参数后，则只有经过 sed 特殊处理的那一行（或者动作）才会被列出来。
-e  ：直接在命令行界面上进行 sed 的动作编辑；
-f  ：直接将 sed 的动作写在一个文件内， -f filename 则可以执行 filename 内的 sed 动作；
-r  ：sed 的动作支持的是延伸型正则表达式的语法。（默认是基础正则表达式语法）
-i  ：直接修改读取的文件内容，而不是由屏幕输出。

动作说明：  [n1[,n2]]function
n1, n2 ：不见得会存在，一般代表"选择进行动作的行数"，举例来说，如果我的动作
         是需要在 10 到 20 行之间进行的，则" 10,20[动作行为] "

function 有下面这些咚咚：
a   ：新增， a 的后面可以接字串，而这些字串会在新的一行出现（目前的下一行）～
c   ：取代， c 的后面可以接字串，这些字串可以取代 n1,n2 之间的行！
d   ：删除，因为是删除啊，所以 d 后面通常不接任何咚咚；
i   ：插入， i 的后面可以接字串，而这些字串会在新的一行出现（目前的上一行）；
p   ：打印，亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运行～
s   ：取代，可以直接进行取代的工作哩！通常这个 s 的动作可以搭配正则表达式！
      例如 1,20s/old/new/g 就是啦！
```

- 以行为单位的新增/删除功能

sed 光是用看的是看不懂的啦！所以又要来练习了！先来玩玩删除与新增的功能吧！

```
范例一：将 /etc/passwd 的内容列出并且打印行号，同时，请将第 2~5 行删除！
[dmtsai@study ~]$ nl /etc/passwd &#124; sed '2,5d'
     1  root:x:0:0:root:/root:/bin/bash
     6  sync:x:5:0:sync:/sbin:/bin/sync
     7  shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
.....(后面省略).....
```

看到了吧？sed 的动作为 '2,5d' ，那个 d 就是删除！因为 2-5 行给他删除了，所以显示的数据就没有 2-5 行啰～ 另外，注意一下，原本应该是要下达 sed -e 才对，没有 -e 也行啦！同时也要注意的是， sed 后面接的动作，请务必以 '' 两个单引号括住喔！

如果题型变化一下，举例来说，如果只要删除第 2 行，可以使用“ nl /etc/passwd | sed '2d' ”来达成， 至于若是要删除第 3 到最后一行，则是“ nl /etc/passwd | sed '3,$d' ”的啦，那个钱字号“ $ ”代表最后一行！

```
范例二：承上题，在第二行后（亦即是加在第三行）加上“drink tea?”字样！
[dmtsai@study ~]$ nl /etc/passwd &#124; sed '2a drink tea'
     1  root:x:0:0:root:/root:/bin/bash
     2  bin:x:1:1:bin:/bin:/sbin/nologin
drink tea
     3  daemon:x:2:2:daemon:/sbin:/sbin/nologin
.....(后面省略).....
```

嘿嘿！在 a 后面加上的字串就已将出现在第二行后面啰！那如果是要在第二行前呢？“ nl /etc/passwd | sed '2i drink tea' ”就对啦！就是将“ a ”变成“ i ”即可。 增加一行很简单，那如果是要增将两行以上呢？

```
范例三：在第二行后面加入两行字，例如“Drink tea or .....”与“drink beer?”
[dmtsai@study ~]$ nl /etc/passwd &#124; sed '2a Drink tea or ......\
&gt; drink beer ?'
     1  root:x:0:0:root:/root:/bin/bash
     2  bin:x:1:1:bin:/bin:/sbin/nologin
Drink tea or ......
drink beer ?
     3  daemon:x:2:2:daemon:/sbin:/sbin/nologin
.....(后面省略).....
```

这个范例的重点是“我们可以新增不只一行喔！可以新增好几行”但是每一行之间都必须要以反斜线“ \ ”来进行新行的增加喔！所以，上面的例子中，我们可以发现在第一行的最后面就有 \ 存在啦！在多行新增的情况下，\ 是一定要的喔！

- 以行为单位的取代与显示功能

刚刚是介绍如何新增与删除，那么如果要整行取代呢？看看下面的范例吧：

```
范例四：我想将第2-5行的内容取代成为“No 2-5 number”呢？
[dmtsai@study ~]$ nl /etc/passwd &#124; sed '2,5c No 2-5 number'
     1  root:x:0:0:root:/root:/bin/bash
No 2-5 number
     6  sync:x:5:0:sync:/sbin:/bin/sync
.....（后面省略）.....
```

通过这个方法我们就能够将数据整行取代了！非常容易吧！sed 还有更好用的东东！我们以前想要列出第 11~20 行， 得要通过“head -n 20 | tail -n 10”之类的方法来处理，很麻烦啦～ sed 则可以简单的直接取出你想要的那几行！是通过行号来捉的喔！看看下面的范例先：

```
范例五：仅列出 /etc/passwd 文件内的第 5-7 行
[dmtsai@study ~]$ nl /etc/passwd &#124; sed -n '5,7p'
     5  lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
     6  sync:x:5:0:sync:/sbin:/bin/sync
     7  shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
```

上述的指令中有个重要的选项“ -n ”，按照说明文档，这个 -n 代表的是“安静模式”！ 那么为什么要使用安静模式呢？你可以自行下达 sed '5,7p' 就知道了 （5-7 行会重复输出）！ 有没有加上 -n 的参数时，输出的数据可是差很多的喔！你可以通过这个 sed 的以行为单位的显示功能， 就能够将某一个文件内的某些行号捉出来查阅！很棒的功能！不是吗？

- 部分数据的搜寻并取代的功能

除了整行的处理模式之外， sed 还可以用行为单位进行部分数据的搜寻并取代的功能喔！ 基本上 sed 的搜寻与取代的与 vi 相当的类似！他有点像这样：

```
sed 's/要被取代的字串/新的字串/g'
```

上表中特殊字体的部分为关键字，请记下来！至于三个斜线分成两栏就是新旧字串的替换啦！ 我们使用下面这个取得 IP 数据的范例，一段一段的来处理给您瞧瞧，让你了解一下什么是咱们所谓的搜寻并取代吧！

```
步骤一：先观察原始讯息，利用 /sbin/ifconfig  查询 IP 为何？
[dmtsai@study ~]$ /sbin/ifconfig eth0
eth0: flags=4163&lt;UP,BROADCAST,RUNNING,MULTICAST&gt;  mtu 1500
        inet 192.168.1.100  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::5054:ff:fedf:e174  prefixlen 64  scopeid 0x20&lt;link&gt;
        ether 52:54:00:df:e1:74  txqueuelen 1000  （Ethernet）
.....（以下省略）.....
# 因为我们还没有讲到 IP ，这里你先有个概念即可啊！我们的重点在第二行，
# 也就是 192.168.1.100 那一行而已！先利用关键字捉出那一行！

步骤二：利用关键字配合 grep 撷取出关键的一行数据
[dmtsai@study ~]$ /sbin/ifconfig eth0 &#124; grep 'inet '
        inet 192.168.1.100  netmask 255.255.255.0  broadcast 192.168.1.255
# 当场仅剩下一行！要注意， CentOS 7 与 CentOS 6 以前的 ifconfig 指令输出结果不太相同，
# 鸟哥这个范例主要是针对 CentOS 7 以后的喔！接下来，我们要将开始到 addr: 通通删除，
# 就是像下面这样：
# inet 192.168.1.100  netmask 255.255.255.0  broadcast 192.168.1.255
# 上面的删除关键在于“ ^.*inet  ”啦！正则表达式出现！ ^_^

步骤三：将 IP 前面的部分予以删除
[dmtsai@study ~]$ /sbin/ifconfig eth0 &#124; grep 'inet ' &#124; sed 's/^.*inet //g'
192.168.1.100  netmask 255.255.255.0  broadcast 192.168.1.255
# 仔细与上个步骤比较一下，前面的部分不见了！接下来则是删除后续的部分，亦即：
192.168.1.100 netmask 255.255.255.0  broadcast 192.168.1.255
# 此时所需的正则表达式为：“ ' *netmask.*$ ”就是啦！

步骤四：将 IP 后面的部分予以删除
[dmtsai@study ~]$ /sbin/ifconfig eth0 &#124; grep 'inet ' &#124; sed 's/^.*inet //g' \
&gt;   &#124; sed 's/ *netmask.*$//g'
192.168.1.100
```

通过这个范例的练习也建议您依据此一步骤来研究你的指令！就是先观察，然后再一层一层的试做， 如果有做不对的地方，就先予以修改，改完之后测试，成功后再往下继续测试。以鸟哥上面的介绍中， 那一大串指令就做了四个步骤！对吧！ ^_^

让我们再来继续研究 sed 与正则表达式的配合练习！假设我只要 MAN 存在的那几行数据，但是含有 # 在内的注解我不想要，而且空白行我也不要！此时该如何处理呢？可以通过这几个步骤来实作看看：

```
步骤一：先使用 grep 将关键字 MAN 所在行取出来
[dmtsai@study ~]$ cat /etc/man_db.conf &#124; grep 'MAN'
# MANDATORY_MANPATH                     manpath_element
# MANPATH_MAP           path_element    manpath_element
# MANDB_MAP             global_manpath  [relative_catpath]
# every automatically generated MANPATH includes these fields
....（后面省略）....

步骤二：删除掉注解之后的数据！
[dmtsai@study ~]$ cat /etc/man_db.conf &#124; grep 'MAN'&#124; sed 's/#.*$//g'

MANDATORY_MANPATH                       /usr/man
....（后面省略）....
# 从上面可以看出来，原本注解的数据都变成空白行啦！所以，接下来要删除掉空白行

[dmtsai@study ~]$ cat /etc/man_db.conf &#124; grep 'MAN'&#124; sed 's/#.*$//g' &#124; sed
MANDATORY_MANPATH                       /usr/man
MANDATORY_MANPATH                       /usr/share/man
MANDATORY_MANPATH                       /usr/local/share/man
....（后面省略）....
```

- 直接修改文件内容（危险动作）

你以为 sed 只有这样的能耐吗？那可不！ sed 甚至可以直接修改文件的内容呢！而不必使用管线命令或数据流重导向！ 不过，由于这个动作会直接修改到原始的文件，所以请你千万不要随便拿系统配置文件来测试喔！ 我们还是使用你下载的 regular_express.txt 文件来测试看看吧！

```
范例六：利用 sed 将 regular_express.txt 内每一行结尾若为 . 则换成 !
[dmtsai@study ~]$ sed -i 's/\.$/\!/g' regular_express.txt
# 上头的 -i 选项可以让你的 sed 直接去修改后面接的文件内容而不是由屏幕输出喔！
# 这个范例是用在取代！请您自行 cat 该文件去查阅结果啰！

范例七：利用 sed 直接在 regular_express.txt 最后一行加入“# This is a test”
[dmtsai@study ~]$ sed -i '$a # This is a test' regular_express.txt
# 由于 $ 代表的是最后一行，而 a 的动作是新增，因此该文件最后新增啰！
```

sed 的“ -i ”选项可以直接修改文件内容，这功能非常有帮助！举例来说，如果你有一个 100 万行的文件，你要在第 100 行加某些文字，此时使用 vim 可能会疯掉！因为文件太大了！那怎办？就利用 sed 啊！通过 sed 直接修改/取代的功能，你甚至不需要使用 vim 去修订！很棒吧！

总之，这个 sed 不错用啦！而且很多的 shell script 都会使用到这个指令的功能～ sed 可以帮助系统管理员管理好日常的工作喔！要仔细的学习呢！

11.3 延伸正则表达式

事实上，一般读者只要了解基础型的正则表达式大概就已经相当足够了，不过，某些时刻为了要简化整个指令操作， 了解一下使用范围更广的延伸型正则表达式的表示式会更方便呢！举个简单的例子好了，在上节的例题三的最后一个例子中，我们要去除空白行与行首为 # 的行列，使用的是

> grep -v '^$' regular_express.txt | grep -v '^#'

需要使用到管线命令来搜寻两次！那么如果使用延伸型的正则表达式，我们可以简化为：

> egrep -v '^$|^#' regular_express.txt

延伸型正则表达式可以通过群组功能“ | ”来进行一次搜寻！那个在单引号内的管线意义为“或 or”啦！ 是否变的更简单呢？此外，grep 默认仅支持基础正则表达式，如果要使用延伸型正则表达式，你可以使用 grep -E ， 不过更建议直接使用 egrep ！直接区分指令比较好记忆！其实 egrep 与 grep -E 是类似命令别名的关系啦！

熟悉了正则表达式之后，到这个延伸型的正则表达式，你应该也会想到，不就是多几个重要的特殊符号吗？ ^^y 是的～所以，我们就直接来说明一下，延伸型正则表达式有哪几个特殊符号？由于下面的范例还是有使用到 *regular_express.txt* ，不巧的是刚刚我们可能将该文件修改过了 @@，所以，请重新下载该文件来练习喔！

RE 字符	意义与范例
+	意义：重复“一个或一个以上”的前一个 RE 字符 范例：搜寻（god）（good）（goood）... 等等的字串。 那个 o+ 代表“一个以上的 o ”所以，下面的执行成果会将第 1, 9, 13 行列出来。 `> egrep -n 'go+d' regular_express.txt`
?	意义：“零个或一个”的前一个 RE 字符 范例：搜寻（gd）（god）这两个字串。 那个 o? 代表“空的或 1 个 o ”所以，上面的执行成果会将第 13, 14 行列出来。 有没有发现到，这两个案例（ 'go+d' 与 'go?d' ）的结果集合与 'go*d' 相同？ 想想看，这是为什么喔！ ^_^ `> egrep -n 'go?d' regular_express.txt`
|	意义：用或（ or ）的方式找出数个字串 范例：搜寻 gd 或 good 这两个字串，注意，是“或”！ 所以，第 1,9,14 这三行都可以被打印出来喔！那如果还想要找出 dog 呢？ `> egrep -n 'gd|good' regular_express.txt` `> egrep -n 'gd|good|dog' regular_express.txt`
( )	意义：找出“群组”字串 范例：搜寻（glad）或（good）这两个字串，因为 g 与 d 是重复的，所以， 我就可以将 la 与 oo 列于（ ）当中，并以 | 来分隔开来，就可以啦！ `> egrep -n 'g（la|oo）d' regular_express.txt`
( )+	意义：多个重复群组的判别 范例：将“AxyzxyzxyzxyzC”用 echo 叫出，然后再使用如下的方法搜寻一下！ `> echo 'AxyzxyzxyzxyzC'	egrep 'A（xyz）+C'`

上面的例子意思是说，我要找开头是 A 结尾是 C ，中间有一个以上的 "xyz" 字串的意思～

以上这些就是延伸型的正则表达式的特殊字符。另外，要特别强调的是，那个！在正则表达式当中并不是特殊字符， 所以，如果你想要查出来文件中含有！与 > 的字行时，可以这样：

```
> grep -n '[!&gt;]' regular_express.txt
```

这样可以了解了吗？常常看到有陷阱的题目写：“反向选择这样对否？ '[!a-z]'？”， 呵呵！是错的呦～要 'a-z 才是对的！至于更多关于正则表达式的进阶文章，请参考文末的参考数据[2]

11.4 文件的格式化与相关处理

接下来让我们来将文件进行一些简单的编排吧！下面这些动作可以将你的讯息进行排版的动作， 不需要重新以 vim 去编辑，通过数据流重导向配合下面介绍的 printf 功能，以及 awk 指令， 就可以让你的讯息以你想要的模样来输出了！试看看吧！

11.4.1 格式化打印： printf

在很多时候，我们可能需要将自己的数据给他格式化输出的！ 举例来说，考试卷分数的输出，姓名与科目及分数之间，总是可以稍微作个比较漂亮的版面配置吧？ 例如我想要输出下面的样式：

```
Name     Chinese   English   Math    Average
DmTsai        80        60     92      77.33
VBird         75        55     80      70.00
Ken           60        90     70      73.33
```

上表的数据主要分成五个字段，各个字段之间可使用 tab 或空白键进行分隔。 请将上表的数据转存成为 printf.txt 文件名，等一下我们会利用这个文件来进行几个小练习的。 因为每个字段的原始数据长度其实并非是如此固定的 （Chinese 长度就是比 Name 要多）， 而我就是想要如此表示出这些数据，此时，就得需要打印格式管理员 printf 的帮忙了！ printf 可以帮我们将数据输出的结果格式化，而且而支持一些特殊的字符～下面我们就来看看！

```
[dmtsai@study ~]$ printf '打印格式' 实际内容
选项与参数：
关于格式方面的几个特殊样式：
       \a    警告声音输出
       \b    倒退键（backspace）
       \f    清除屏幕 （form feed）
       \n    输出新的一行
       \r    亦即 Enter 按键
       \t    水平的 [tab] 按键
       \v    垂直的 [tab] 按键
       \xNN  NN 为两位数的数字，可以转换数字成为字符。
关于 C 程序语言内，常见的变量格式
       %ns   那个 n 是数字， s 代表 string ，亦即多少个字符；
       %ni   那个 n 是数字， i 代表 integer ，亦即多少整数码数；
       %N.nf 那个 n 与 N 都是数字， f 代表 floating （浮点），如果有小数码数，
             假设我共要十个位数，但小数点有两位，即为 %10.2f 啰！
```

接下来我们来进行几个常见的练习。假设所有的数据都是一般文字 （这也是最常见的状态），因此最常用来分隔数据的符号就是 [Tab] 啦！因为 [Tab] 按键可以将数据作个整齐的排列！那么如何利用 printf 呢？参考下面这个范例：

```
范例一：将刚刚上头数据的文件 （printf.txt） 内容仅列出姓名与成绩：（用 [tab] 分隔）
[dmtsai@study ~]$ printf '%s\t %s\t %s\t %s\t %s\t \n' $（cat printf.txt）
Name      Chinese         English         Math     Average
DmTsai    80       60       92       77.33
VBird     75       55       80       70.00
Ken       60       90       70       73.33
```

由于 printf 并不是管线命令，因此我们得要通过类似上面的功能，将文件内容先提出来给 printf 作为后续的数据才行。 如上所示，我们将每个数据都以 [tab] 作为分隔，但是由于 Chinese 长度太长，导致 English 中间多了一个 [tab] 来将数据排列整齐！啊～结果就看到数据对齐结果的差异了！

另外，在 printf 后续的那一段格式中，%s 代表一个不固定长度的字串，而字串与字串中间就以 \t 这个 [tab] 分隔符号来处理！你要记得的是，由于 \t 与 %s 中间还有空格，因此每个字串间会有一个 [tab] 与一个空白键的分隔喔！

既然每个字段的长度不固定会造成上述的困扰，那我将每个字段固定就好啦！没错没错！这样想非常好！ 所以我们就将数据给他进行固定字段长度的设计吧！

```
范例二：将上述数据关于第二行以后，分别以字串、整数、小数点来显示：
[dmtsai@study ~]$ printf '%10s %5i %5i %5i %8.2f \n' $（cat printf.txt &#124; grep -v Nam
    DmTsai    80    60    92    77.33
     VBird    75    55    80    70.00
       Ken    60    90    70    73.33
```

上面这一串格式想必您看得很辛苦！没关系！一个一个来解释！上面的格式共分为五个字段， %10s 代表的是一个长度为 10 个字符的字串字段，%5i 代表的是长度为 5 个字符的数字字段，至于那个 %8.2f 则代表长度为 8 个字符的具有小数点的字段，其中小数点有两个字符宽度。我们可以使用下面的说明来介绍 %8.2f 的意义：

> 字符宽度： 12345678 %8.2f意义：00000.00

如上所述，全部的宽度仅有 8 个字符，整数部分占有 5 个字符，小数点本身 （.） 占一位，小数点下的位数则有两位。 这种格式经常使用于数值程序的设计中！这样了解乎？自己试看看如果要将小数点位数变成 1 位又该如何处理？

printf 除了可以格式化处理之外，他还可以依据 ASCII 的数字与图形对应来显示数据喔[3]！举例来说 16 进位的 45 可以得到什么 ASCII 的显示图 （其实是字符啦）？

```
范例三：列出 16 进位数值 45 代表的字符为何？
[dmtsai@study ~]$ printf '\x45\n'
E
# 这东西也很好玩～他可以将数值转换成为字符，如果你会写 script 的话，
# 可以自行测试一下，由 20~80 之间的数值代表的字符是啥喔！ ^_^
```

printf 的使用相当的广泛喔！包括等一下后面会提到的 awk 以及在 C 程序语言当中使用的屏幕输出， 都是利用 printf 呢！鸟哥这里也只是列出一些可能会用到的格式而已，有兴趣的话，可以自行多作一些测试与练习喔！ ^_^



Tips 打印格式化这个 printf 指令，乍看之下好像也没有什么很重要的～ 不过，如果你需要自行撰写一些软件，需要将一些数据在屏幕上头漂漂亮亮的输出的话， 那么 printf 可也是一个很棒的工具喔！

11.4.2 awk：好用的数据处理工具

awk 也是一个非常棒的数据处理工具！相较于 sed 常常作用于一整个行的处理， awk 则比较倾向于一行当中分成数个“字段”来处理。因此，awk 相当的适合处理小型的数据数据处理呢！awk 通常运行的模式是这样的：

```
[dmtsai@study ~]$ awk '条件类型1{动作1} 条件类型2{动作2} ...' filename
```

awk 后面接两个单引号并加上大括号 {} 来设置想要对数据进行的处理动作。 awk 可以处理后续接的文件，也可以读取来自前个指令的 standard output 。 但如前面说的， awk 主要是处理“每一行的字段内的数据”，而默认的“字段的分隔符号为 "空白键" 或 "[tab]键" ”！举例来说，我们用 last 可以将登陆者的数据取出来，结果如下所示：

```
[dmtsai@study ~]$ last -n 5 &lt;==仅取出前五行
dmtsai   pts/0     192.168.1.100   Tue Jul 14 17:32   still logged in
dmtsai   pts/0     192.168.1.100   Thu Jul  9 23:36 - 02:58  (03:22)
dmtsai   pts/0     192.168.1.100   Thu Jul  9 17:23 - 23:36  (06:12)
dmtsai   pts/0     192.168.1.100   Thu Jul  9 08:02 - 08:17  (00:14)
dmtsai   tty1                      Fri May 29 11:55 - 12:11  (00:15)
```

若我想要取出帐号与登陆者的 IP ，且帐号与 IP 之间以 [tab] 隔开，则会变成这样：

```
[dmtsai@study ~]$ last -n 5 &#124; awk '{print $1 "\t" $3}'
dmtsai  192.168.1.100
dmtsai  192.168.1.100
dmtsai  192.168.1.100
dmtsai  192.168.1.100
dmtsai  Fri
```

上表是 awk 最常使用的动作！通过 print 的功能将字段数据列出来！字段的分隔则以空白键或 [tab] 按键来隔开。 因为不论哪一行我都要处理，因此，就不需要有 "条件类型" 的限制！我所想要的是第一栏以及第三栏， 但是，第五行的内容怪怪的～这是因为数据格式的问题啊！所

以啰～使用 awk 的时候，请先确认一下你的数据当中，如果是连续性的数据，请不要有空格或 [tab] 在内，否则，就会像这个例子这样，会发生误判喔！

另外，由上面这个例子你也会知道，在 awk 的括号内，每一行的每个字段都是有变量名称的，那就是 $1, $2... 等变量名称。以上面的例子来说， dmtsai 是 $1 ，因为他是第一栏嘛！至于 192.168.1.100 是第三栏， 所以他就是 $3 啦！后面以此类推～呵呵！还有个变量喔！那就是 $0 ，$0 代表“一整列数据”的意思～以上面的例子来说，第一行的 $0 代表的就是“dmtsai .... ”那一行啊！ 由此可知，刚刚上面五行当中，整个 awk 的处理流程是：

1. 读入第一行，并将第一行的数据填入 $0, $1, $2.... 等变量当中；
2. 依据 "条件类型" 的限制，判断是否需要进行后面的 "动作"；
3. 做完所有的动作与条件类型；
4. 若还有后续的“行”的数据，则重复上面 1~3 的步骤，直到所有的数据都读完为止。

经过这样的步骤，你会晓得， awk 是“以行为一次处理的单位”， 而“以字段为最小的处理单位”。好了，那么 awk 怎么知道我到底这个数据有几行？有几栏呢？这就需要 awk 的内置变量的帮忙啦～

变量名称	代表意义
NF	每一行 （$0） 拥有的字段总数
NR	目前 awk 所处理的是“第几行”数据
FS	目前的分隔字符，默认是空白键

我们继续以上面 last -n 5 的例子来做说明，如果我想要：

- 列出每一行的帐号（就是 $1）；
- 列出目前处理的行数（就是 awk 内的 NR 变量）
- 并且说明，该行有多少字段（就是 awk 内的 NF 变量）

则可以这样：



Tips 要注意喔，awk 后续的所有动作是以单引号“ ' ”括住的，由于单引号与双引号都必须是成对的， 所以， awk 的格式内容如果想要以 print 打印时，记得非变量的文字部分，包含上一小节 printf 提到的格式中，都需要使用双引号来定义出来喔！因为单引号已经是 awk 的指令固定用法了！

```
[dmtsai@study ~]$ last -n 5&#124; awk '{print $1 "\t lines: " NR "\t columns: " NF}'
dmtsai   lines: 1        columns: 10
dmtsai   lines: 2        columns: 10
dmtsai   lines: 3        columns: 10
dmtsai   lines: 4        columns: 10
dmtsai   lines: 5        columns: 9
# 注意喔，在 awk 内的 NR, NF 等变量要用大写，且不需要有钱字号 $ 啦！
```

这样可以了解 NR 与 NF 的差别了吧？好了，下面来谈一谈所谓的 "条件类型" 了吧！

- awk 的逻辑运算字符

既然有需要用到 "条件" 的类别，自然就需要一些逻辑运算啰～例如下面这些：

运算单元	代表意义
>	大于
<	小于
>=	大于或等于
<=	小于或等于
==	等于
!=	不等于

值得注意的是那个“ == ”的符号，因为：

- 逻辑运算上面亦即所谓的大于、小于、等于等判断式上面，习惯上是以“ == ”来表示；
- 如果是直接给予一个值，例如变量设置时，就直接使用 = 而已。

好了，我们实际来运用一下逻辑判断吧！举例来说，在 /etc/passwd 当中是以冒号 ":" 来作为字段的分隔， 该文件中第一字段为帐号，第三字段则是 UID。那假设我要查阅，第三栏小于 10 以下的数据，并且仅列出帐号与第三栏， 那么可以这样做：

```
[dmtsai@study ~]$ cat /etc/passwd &#124; awk '{FS=":"} $3 &lt; 10 {print $1 "\t " $3}'
root:x:0:0:root:/root:/bin/bash
bin      1
daemon   2
....（以下省略）....
```

有趣吧！不过，怎么第一行没有正确的显示出来呢？这是因为我们读入第一行的时候，那些变量 $1, $2... 默认还是以空白键为分隔的，所以虽然我们定义了 FS=":" 了， 但是却仅能在第二行后才开始生效。那么怎么办呢？我们可以预先设置 awk 的变量啊！ 利用 BEGIN 这个关键字喔！这样做：

```
[dmtsai@study ~]$ cat /etc/passwd &#124; awk 'BEGIN {FS=":"} $3 &lt; 10 {print $1 "\t " $
root      0
bin       1
daemon    2
......（以下省略）......
```

很有趣吧！而除了 BEGIN 之外，我们还有 END 呢！另外，如果要用 awk 来进行“计算功能”呢？以下面的例子来看， 假设我有一个薪资数据表文件名为 pay.txt ，内容是这样的：

```
Name    1st     2nd     3th
VBird   23000   24000   25000
DMTsai  21000   20000   23000
Bird2   43000   42000   41000
```

如何帮我计算每个人的总额呢？而且我还想要格式化输出喔！我们可以这样考虑：

- 第一行只是说明，所以第一行不要进行加总 （NR==1 时处理）；
- 第二行以后就会有加总的情况出现 （NR>=2 以后处理）

```
[dmtsai@study ~]$ cat pay.txt &#124; \
&gt; awk 'NR==1{printf "%10s %10s %10s %10s %10s\n",$1,$2,$3,$4,"Total" }
&gt; NR&gt;=2{total = $2 + $3 + $4
&gt; printf "%10s %10d %10d %10d %10.2f\n", $1, $2, $3, $4, total}'
      Name        1st        2nd        3th      Total
     VBird      23000      24000      25000   72000.00
    DMTsai      21000      20000      23000   64000.00
     Bird2      43000      42000      41000  126000.00
```

上面的例子有几个重要事项应该要先说明的：

- awk 的指令间隔：所有 awk 的动作，亦即在 {} 内的动作，如果有需要多个指令辅助时，可利用分号“;”间隔， 或者直接以 [Enter] 按键来隔开每个指令，例如上面的范例中，鸟哥共按了三次 [enter] 喔！
- 逻辑运算当中，如果是“等于”的情况，则务必使用两个等号“==”！
- 格式化输出时，在 printf 的格式设置当中，务必加上 \n ，才能进行分行！
- 与 bash shell 的变量不同，在 awk 当中，变量可以直接使用，不需加上 $ 符号。

利用 awk 这个玩意儿，就可以帮我们处理很多日常工作了呢！真是好用的很～ 此外， awk 的输出格式当中，常常会以 printf 来辅助，所以， 最好你对 printf 也稍微熟悉一下比较好啦！另外， awk 的动作内 {} 也是支持 if （条件） 的喔！ 举例来说，上面的指令可以修订成为这样：

```
[dmtsai@study ~]$ cat pay.txt &#124; \
&gt; awk '{if（NR==1） printf "%10s %10s %10s %10s %10s\n",$1,$2,$3,$4,"Total"}
&gt; NR&gt;=2{total = $2 + $3 + $4
&gt; printf "%10s %10d %10d %10d %10.2f\n", $1, $2, $3, $4, total}'
```

你可以仔细的比对一下上面两个输入有啥不同～从中去了解两种语法吧！我个人是比较倾向于使用第一种语法， 因为会比较有统一性啊！ ^_^

除此之外， awk 还可以帮我们进行循环计算喔！真是相当的好用！不过，那属于比较进阶的单独课程了， 我们这里就不再多加介绍。如果你有兴趣的话，请务必参考延伸阅读中的相关链接喔 [4]。

11.4.3 文件比对工具

什么时候会用到文件的比对啊？通常是“同一个套装软件的不同版本之间，比较配置文件与原始文件的差异”。 很多时候所谓的文件比对，通常是用在 ASCII 纯文本文件的比对上的！那么比对文件的指令有哪些？最常见的就是 diff 啰！ 另外，除了 diff 比对之外，我们还可以借由 cmp 来比对非纯文本文件！同时，也能够借由 diff 创建的分析档， 以处理补丁 （patch） 功能的文件呢！就来玩玩先！

- diff

diff 就是用在比对两个文件之间的差异的，并且是以行为单位来比对的！一般是用在 ASCII 纯文本文件的比对上。 由于是以行为比对的单位，因此 diff 通常是用在同一的文件（或软件）的新旧版本差异上！ 举例来说，假如我们要将 /etc/passwd 处理成为一个新的版本，处理方式为： 将第四行删除，第六行则取代成为“no six line”，新的文件放置到 /tmp/test 里面，那么应该怎么做？

```
[dmtsai@study ~]$ mkdir -p /tmp/testpw &lt;==先创建测试用的目录
[dmtsai@study ~]$ cd /tmp/testpw
[dmtsai@study testpw]$ cp /etc/passwd passwd.old
[dmtsai@study testpw]$ cat /etc/passwd &#124; sed -e '4d' -e '6c no six line' &gt; passwd
# 注意一下， sed 后面如果要接超过两个以上的动作时，每个动作前面得加 -e 才行！
# 通过这个动作，在 /tmp/testpw 里面便有新旧的 passwd 文件存在了！
```

接下来讨论一下关于 diff 的用法吧！

```
[dmtsai@study ~]$ diff [-bBi] from-file to-file
选项与参数：
from-file ：一个文件名，作为原始比对文件的文件名；
to-file   ：一个文件名，作为目的比对文件的文件名；
注意，from-file 或 to-file 可以 - 取代，那个 - 代表“Standard input”之意。

-b  ：忽略一行当中，仅有多个空白的差异(例如 "about me" 与 "about     me" 视为相同
-B  ：忽略空白行的差异。
-i  ：忽略大小写的不同。

范例一：比对 passwd.old 与 passwd.new 的差异：
[dmtsai@study testpw]$ diff passwd.old passwd.new
4d3    &lt;==左边第四行被删除 （d） 掉了，基准是右边的第三行
&lt; adm:x:3:4:adm:/var/adm:/sbin/nologin  &lt;==这边列出左边(&lt;)文件被删除的那一行内容
6c5    &lt;==左边文件的第六行被取代 （c） 成右边文件的第五行
&lt; sync:x:5:0:sync:/sbin:/bin/sync  &lt;==左边(&lt;)文件第六行内容
---
&gt; no six line                     &lt;==右边(&gt;)文件第五行内容
# 很聪明吧！用 diff 就把我们刚刚的处理给比对完毕了！
```

用 diff 比对文件真的是很简单喔！不过，你不要用 diff 去比对两个完全不相干的文件，因为比不出个啥咚咚！ 另外， diff 也可以比对整个目录下的差异喔！举例来说，我们想要了解一下不同的开机执行等级 （runlevel） 内容有啥不同？假设你已经知道执行等级 0 与 5 的启动脚本分别放置到 /etc/rc0.d 及 /etc/rc5.d ， 则我们可以将两个目录比对一下：

```
[dmtsai@study ~]$ diff /etc/rc0.d/ /etc/rc5.d/
Only in /etc/rc0.d/: K90network
Only in /etc/rc5.d/: S10network
```

我们的 diff 很聪明吧！还可以比对不同目录下的相同文件名的内容，这样真的很方便喔～

- cmp

相对于 diff 的广泛用途， cmp 似乎就用的没有这么多了～ cmp 主要也是在比对两个文件，他主要利用“字节”单位去比对， 因此，当然也可以比对 binary file 啰～（还是要再提醒喔， diff 主要是以“行”为单位比对， cmp 则是以“字节”为单位去比对，这并不相同！）

```
[dmtsai@study ~]$ cmp [-l] file1 file2
选项与参数：
-l  ：将所有的不同点的字节处都列出来。因为 cmp 默认仅会输出第一个发现的不同点。

范例一：用 cmp 比较一下 passwd.old 及 passwd.new
[dmtsai@study testpw]$ cmp passwd.old passwd.new
passwd.old passwd.new differ: char 106, line 4
```

看到了吗？第一个发现的不同点在第四行，而且字节数是在第 106 个字节处！这个 cmp 也可以用来比对 binary 啦！ ^_^

- patch

patch 这个指令与 diff 可是有密不可分的关系啊！我们前面提到，diff 可以用来分辨两个版本之间的差异， 举例来说，刚刚我们所创建的 passwd.old 及 passwd.new 之间就是两个不同版本的文件。 那么，如果要“升级”呢？就是“将旧的文件升级成为新的文件”时，应该要怎么做呢？ 其实也不难啦！就是“先比较先旧版本的差异，并将差异档制作成为补丁文件，再由补丁文件更新旧文件”即可。 举例来说，我们可以这样做测试：

```
范例一：以 /tmp/testpw 内的 passwd.old 与 passwd.new  制作补丁文件
[dmtsai@study testpw]$ diff -Naur passwd.old passwd.new &gt; passwd.patch
[dmtsai@study testpw]$ cat passwd.patch
--- passwd.old  2015-07-14 22:37:43.322535054 +0800  &lt;==新旧文件的信息
+++ passwd.new  2015-07-14 22:38:03.010535054 +0800
@@ -1,9 +1,8 @@  &lt;==新旧文件要修改数据的界定范围，旧文件在 1-9 行，新文件在 1-8 行
 root:x:0:0:root:/root:/bin/bash
 bin:x:1:1:bin:/bin:/sbin/nologin
 daemon:x:2:2:daemon:/sbin:/sbin/nologin
-adm:x:3:4:adm:/var/adm:/sbin/nologin     &lt;==左侧文件删除
 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
-sync:x:5:0:sync:/sbin:/bin/sync          &lt;==左侧文件删除
+no six line                             &lt;==右侧新文件加入
 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
 halt:x:7:0:halt:/sbin:/sbin/halt
 mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
```

一般来说，使用 diff 制作出来的比较文件通常使用扩展名为 .patch 啰。至于内容就如同上面介绍的样子。 基本上就是以行为单位，看看哪边有一样与不一样的，找到一样的地方，然后将不一样的地方取代掉！ 以上面表格为例，新文件看到 - 会删除，看到 + 会加入！好了，那么如何将旧的文件更新成为新的内容呢？ 就是将 passwd.old 改成与 passwd.new 相同！可以这样做：

```
# 因为 CentOS 7 默认没有安装 patch 这个软件，因此得要依据之前介绍的方式来安装一下软件！
# 请记得拿出原本光盘并放入光驱当中，这时才能够使用下面的方式来安装软件！
[dmtsai@study ~]$ su -
[root@study ~]# mount /dev/sr0 /mnt
[root@study ~]# rpm -ivh /mnt/Packages/patch-2.*
[root@study ~]# umount /mnt
[root@study ~]# exit
# 通过上述的方式可以安装好所需要的软件，且无须上网。接下来让我们开始操作 patch 啰！

[dmtsai@study ~]$ patch -pN &lt; patch_file    &lt;==更新
[dmtsai@study ~]$ patch -R -pN &lt; patch_file &lt;==还原
选项与参数：
-p  ：后面可以接“取消几层目录”的意思。
-R  ：代表还原，将新的文件还原成原来旧的版本。

范例二：将刚刚制作出来的 patch file 用来更新旧版数据
[dmtsai@study testpw]$ patch -p0 &lt; passwd.patch
patching file passwd.old
[dmtsai@study testpw]$ ll passwd*
-rw-rw-r--. 1 dmtsai dmtsai 2035 Jul 14 22:38 passwd.new
-rw-r--r--. 1 dmtsai dmtsai 2035 Jul 14 23:30 passwd.old  &lt;==文件一模一样！

范例三：恢复旧文件的内容
[dmtsai@study testpw]$ patch -R -p0 &lt; passwd.patch
[dmtsai@study testpw]$ ll passwd*
-rw-rw-r--. 1 dmtsai dmtsai 2035 Jul 14 22:38 passwd.new
-rw-r--r--. 1 dmtsai dmtsai 2092 Jul 14 23:31 passwd.old
# 文件就这样恢复成为旧版本啰
```

为什么这里会使用 -p0 呢？因为我们在比对新旧版的数据时是在同一个目录下， 因此不需要减去目录啦！如果是使用整体目录比对 （diff 旧目录 新目录） 时， 就得要依据创建 patch 文件所在目录来进行目录的删减啰！

更详细的 patch 用法我们会在后续的第五篇的源代码编译 （第二十一章）再跟大家介绍， 这里仅是介绍给你，我们可以利用 diff 来比对两个文件之间的差异， 更可进一步利用这个功能来制作修补文件 （patch file） ，让大家更容易进行比对与升级呢！很不赖吧！ ^_^

11.4.4 文件打印准备： pr

如果你曾经使用过一些图形接口的文书处理软件的话，那么很容易发现，当我们在打印的时候， 可以同时选择与设置每一页打印时的标头吧！也可以设置页码呢！那么，如果我是在 Linux 下面打印纯文本文件呢 可不可以具有标题啊？可不可以加入页码啊？呵呵！当然可以啊！使用 pr 就能够达到这个功能了。不过， pr 的参数实在太多了，鸟哥也说不完，一般来说，鸟哥都仅使用最简单的方式来处理而已。举例来说，如果想要打印 /etc/man_db.conf 呢？

```
[dmtsai@study ~]$ pr /etc/man_db.conf

2014-06-10 05:35                 /etc/man_db.conf                 Page 1

#
#
# This file is used by the man-db package to configure the man and cat paths.
# It is also used to provide a manpath for those without one by examining
# configure script.
.....（以下省略）......
```

上面特殊字体那一行呢，其实就是使用 pr 处理后所造成的标题啦！标题中会有“文件时间”、“文件文件名”及“页码”三大项目。 更多的 pr 使用，请参考 pr 的说明啊！ ^_^

11.5 重点回顾

- 正则表达式就是处理字串的方法，他是以行为单位来进行字串的处理行为；
- 正则表达式通过一些特殊符号的辅助，可以让使用者轻易的达到“搜寻/删除/取代”某特定字串的处理程序；
- 只要工具程序支持正则表达式，那么该工具程序就可以用来作为正则表达式的字串处理之用；
- 正则表达式与万用字符是完全不一样的东西！万用字符 （wildcard） 代表的是 bash 操作接口的一个功能， 但正则表达式则是一种字串处理的表示方式！
- 使用 grep 或其他工具进行正则表达式的字串比对时，因为编码的问题会有不同的状态，因此， 你最好将 LANG 等变量设置为 C 或者是 en 等英文语系！
- grep 与 egrep 在正则表达式里面是很常见的两支程序，其中， egrep 支持更严谨的正则表达式的语法；
- 由于编码系统的不同，不同的语系 （LANG） 会造成正则表达式撷取数据的差异。因此可利用特殊符号如 [:upper:] 来替代编码范围较佳；
- 由于严谨度的不同，正则表达式之上还有更严谨的延伸正则表达式；
- 基础正则表达式的特殊字符有： *, ., [], [-], , ^, $ 等！
- 常见的支持正则表达式的工具软件有： grep , sed, vim 等等
- printf 可以通过一些特殊符号来将数据进行格式化输出；
- awk 可以使用“字段”为依据，进行数据的重新整理与输出；
- 文件的比对中，可利用 diff 及 cmp 进行比对，其中 diff 主要用在纯文本方面的新旧版本比对
- patch 指令可以将旧版数据更新到新版 （主要亦由 diff 创建 patch 的补丁来源文件）

11.6 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ）

- 情境仿眞题一：通过 grep 搜寻特殊字串，并配合数据流重导向来处理大量的文件搜寻问题。
    - 目标：正确的使用正则表达式；
    - 前提：需要了解数据流重导向，以及通过子指令 $（command） 来处理文件名的搜寻；我们简单的以搜寻星号 （*） 来处理下面的任务：
    - 利用正则表达式找出系统中含有某些特殊关键字的文件，举例来说，找出在 /etc 下面含有星号 （*） 的文件与内容：

      解决的方法必须要搭配万用字符，但是星号本身就是正则表达式的字符，因此需要如此进行：

      ```
      [dmtsai@study ~]$ grep '\*' /etc/* 2> /dev/null
      ```

      你必须要注意的是，在单引号内的星号是正则表达式的字符，但我们要找的是星号，因此需要加上跳脱字符 （\）。但是在 /etc/ 的那个 则是 bash 的万用字符！ 代表的是文件的文件名喔！不过由上述的这个结果中，我们仅能找到 /etc 下面第一层子目录的数据，无法找到次目录的数据， 如果想要连同完整的 /etc 次目录数据，就得要这样做：

      ```
      [dmtsai@study ~]$ grep '\*' $（find /etc -type f ） 2> /dev/null
      # 如果只想列出文件名而不要列出内容的话，使用下面的方式来处理即可喔！
      [dmtsai@study ~]$ grep -l '\*' $（find /etc -type f ） 2> /dev/null
      ```

    - 但如果文件数量太多呢？如同上述的案例，如果要找的是全系统 （/） 呢？你可以这样做：

      ```
      [dmtsai@study ~]$ grep '\*' $（find / -type f 2> /dev/null ）
      -bash: /usr/bin/grep: Argument list too long
      ```

      眞要命！由于命令行的内容长度是有限制的，因此当搜寻的对象是整个系统时，上述的指令会发生错误。那该如何是好？ 此时我们可以通过管线命令以及 xargs 来处理。举例来说，让 grep 每次仅能处理 10 个文件名，此时你可以这样想：

      1. 先用 find 去找出文件；
      2. 用 xargs 将这些文件每次丢 10 个给 grep 来作为参数处理；
      3. grep 实际开始搜寻文件内容。 所以整个作法就会变成这样：

```
[dmtsai@study ~]$ find / -type f 2&gt; /dev/null &#124; xargs -n 10 grep '\*'
```

- 从输出的结果来看，数据量实在非常庞大！那如果我只是想要知道文件名而已呢？你可以通过 grep 的功能来找到如下的参数！

```
[dmtsai@study ~]$ find / -type f 2&gt; /dev/null &#124; xargs -n 10 grep -l '\*'
```

- 情境仿真题二：使用管线命令配合正则表达式创建新指令与新变量。我想要创建一个新的指令名为 myip ， 这个指令能够将我系统的 IP 捉出来显示。而我想要有个新变量，变量名为 MYIP ，这个变量可以记录我的 IP 。

  处理的方式很简单，我们可以这样试看看：

  1. 首先，我们依据本章内的 ifconfig, sed 与 awk 来取得我们的 IP ，指令为：

```
[dmtsai@study ~]$ ifconfig eth0 &#124; grep 'inet ' &#124; sed 's/^.*inet //g'&#
```

  2. 再来，我们可以将此指令利用 alias 指定为 myip 喔！如下所示：

```
[dmtsai@study ~]$ alias myip="ifconfig eth0 &#124; grep 'inet ' &#124; sed 's/^.
&gt;   sed 's/ *netmask.*$//g'
```

  3. 最终，我们可以通过变量设置来处理 MYIP 喔！

```
[dmtsai@study ~]$ MYIP=$（ myip ）
```

  4. 如果每次登陆都要生效，可以将 alias 与 MYIP 的设置那两行，写入你的 ~/.bashrc 即可！

简答题部分：

- 我想要知道，在 /etc 下面，只要含有 XYZ 三个字符的任何一个字符的那一行就列出来，要怎样进行？grep [XYZ] /etc/*
- 将 /etc/kdump.conf 内容取出后，（1）去除开头为 # 的行 （2）去除空白行 （3）取出开头为英文字母的那几行 （4）最终统计总行数该如何进行？grep -v '^#' /etc/kdump.conf | grep -v '^$' | grep '^[[:alpha:]]' | wc -l

11.7 参考资料与延伸阅读

- [1]关于正则表达式与 POSIX 及特殊语法的参考网址可以查询下面的来源： 维基百科的说明：http://en.wikipedia.org/wiki/Regular_expression ZYTRAX 网站介绍：http://zytrax.com/tech/web/regex.htm
- [2]其他关于正则表达式的网站介绍： 洪朝贵老师的网页：http://www.cyut.edu.tw/~ckhung/b/re/index.php 龙门少尉的窝：http://main.rtfiber.com.tw/~changyj/ PCRE 官方网站：http://perldoc.perl.org/perlre.html
- [3]关于 ASCII 编码对照表可参考维基百科的介绍： 维基百科 （ASCII） 条目：http://zh.wikipedia.org/w/index.php?title=ASCII&variant=zh-cn
- [4]关于 awk 的进阶文献，包括有下面几个链接： 中研院计算中心 ASPAC 计划之 awk 程序介绍： 鸟哥备份：http://linux.vbird.org/linux_basic/0330regularex/awk.pdf 这份文件写的非常棒！欢迎大家多多参考！ Study Area：http://www.study-area.org/linux/system/linux_shell.htm

2002/07/29：第一次完成； 2003/02/10：重新编排与加入 FAQ ； 2005/01/28：重新汇整基础正则表达式的内容！重点在 regular_express.txt 的处理与练习上！ 2005/03/30：修订了 grep -n 'goog' *regular_express.txt* 这一段 *2005/05/23*：修订了 *grep -n '^[a-z]' regular_express.txt* 所要撷取的是小写，之前写成大写，错了！ *2005/08/22*：加入了 *awk, sed* 等工具的介绍，还有 *diff* 与 *cmp* 等指令的说明！ *2005/09/05*：加入 *printf* 内，关于 *\xNN* 的说明！ *2006/03/10*：将原本的 *sed* 内的动作（*action*）中， *s* 由“搜寻”改成“取代”了！ *2006/10/05*：在 *sed* 当中多了一个 *-i* 的参数说明，也多了一个范例八可以参考。感谢讨论区的*thyme*兄！ *2008/10/08*：加入 *grep* 内的 *--color=auto* 说明！ *2009/02/07*：将旧的基于 *FC4* 版本的文章移动到此处 *2009/02/10*：重新排版，并且加入语系的说明，以及特殊 *[:数据:]* 的说明！更改不少范例的说明。 *2009/05/14*：感谢网友 *Jack* 的回报， *cmp* 应该是使用“字节 *Bytes*”而非位 *bits*，感谢 *Jack* 兄。 *2009/08/26*：加入情境仿真题目了！ *2010/04/16*：由*linux_task*兄提供的意见， 将*[*原本的 说明订正](../Text/index.html#20100416)一些部分，可读性较佳！感谢您！ 2015/07/10：将旧的基于 CentOS 5.x 的版本移动到 这里了！ 2015/07/14：大部分改以一般帐号的身份来操作系统了！不过改动的幅度不大！

第十二章、学习 Shell Scripts

最近更新日期：20//

如果你真的很想要走信息这条路，并且想要管理好属于你的主机，那么，别说鸟哥不告诉你， 可以自动管理系统的好工具： Shell scripts！这家伙真的是得要好好学习学习的！ 基本上， shell script 有点像是早期的批处理文件，亦即是将一些指令汇整起来一次执行，但是 Shell script 拥有更强大的功能，那就是他可以进行类似程序 （program） 的撰写，并且不需要经过编译 （compile） 就能够执行， 真的很方便。加上我们可通过 shell script 来简化我们日常的工作管理， 而且，整个 Linux 环境中，一些服务 （services） 的启动都是通过 shell script 的， 如果你对于 script 不了解，嘿嘿！发生问题时，可真是会求助无门喔！所以，好好的学一学他吧！

12.1 什么是 Shell scripts

什么是 shell script （程序化脚本） 呢？就字面上的意义，我们将他分为两部份。 在“ shell ”部分，我们在 十章的 BASH 当中已经提过了，那是一个命令行下面让我们与系统沟通的一个工具接口。那么“ script ”是啥？ 字面上的意义， script 是“脚本、剧本”的意思。整句话是说，shell script 是针对 shell 所写的“剧本！”

什么东西啊？其实， shell script 是利用 shell 的功能所写的一个“程序 （program）”，这个程序是使用纯文本文件，将一些 shell 的语法与指令（含外部指令）写在里面， 搭配正则表达式、管线命令与数据流重导向等功能，以达到我们所想要的处理目的。

所以，简单的说， shell script 就像是早期 DOS 年代的批处理文件 （.bat） ，最简单的功能就是将许多指令汇整写在一起， 让使用者很轻易的就能够 one touch 的方法去处理复杂的动作 （执行一个文件 "shell script" ，就能够一次执行多个指令）。 而且 shell script 更提供阵列、循环、条件与逻辑判断等重要功能，让使用者也可以直接以 shell 来撰写程序，而不必使用类似 C 程序语言等传统程序撰写的语法呢！

这么说你可以了解了吗？是的！ shell script 可以简单的被看成是批处理文件， 也可以被说成是一个程序语言，且这个程序语言由于都是利用 shell 与相关工具指令， 所以不需要编译即可执行，且拥有不错的除错 （debug） 工具，所以，他可以帮助系统管理员快速的管理好主机。

12.1.1 干嘛学习 shell scripts

这是个好问题：“我又干嘛一定要学 shell script ？我又不是信息人，没有写程序的概念， 那我干嘛还要学 shell script 呢？不要学可不可以啊？”呵呵～如果 Linux 对你而言， 你只是想要“会用”而已，那么，不需要学 shell script 也还无所谓，这部分先给他跳过去， 等到有空的时候，再来好好的瞧一瞧。但是，如果你是真的想要玩清楚 Linux 的来龙去脉， 那么 shell script 就不可不知，为什么呢？因为：

- 自动化管理的重要依据

不用鸟哥说你也知道，管理一部主机真不是件简单的事情，每天要进行的任务就有：查询登录文件、追踪流量、监控使用者使用主机状态、主机各项硬件设备状态、 主机软件更新查询、更不要说得应付其他使用者的突然要求了。而这些工作的进行可以分为： （1） 自行手动处理，或是 （2） 写个简单的程序来帮你每日“自动处理分析”这两种方式，你觉得哪种方式比较好？ 当然是让系统自动工作比较好，对吧！呵呵～这就得要良好的 shell script 来帮忙的啦！

- 追踪与管理系统的重要工作

虽然我们还没有提到服务启动的方法，不过，这里可以先提一下，我们 CentOS 6.x 以前的版本中，系统的服务 （services） 启动的接口是在 /etc/init.d/ 这个目录下，目录下的所有文件都是 scripts ； 另外，包括开机 （booting） 过程也都是利用 shell script 来帮忙搜寻系统的相关设置数据， 然后再代入各个服务的设置参数啊！举例来说，如果我们想要重新启动系统登录文件， 可以使用："/etc/init.d/rsyslogd restart"，那个 rsyslogd 文件就是 script 啦！

另外，鸟哥曾经在某一代的 Fedora 上面发现，启动 MySQL 这个数据库服务时，确实是可以启动的， 但是屏幕上却老是出现"failure"！后来才发现，原来是启动 MySQL 那个 script 会主动的以"空的密码"去尝试登陆 MySQL ，但为了安全性鸟哥修改过 MySQL 的密码啰～当然就登陆失败～ 后来改了改 script ，就略去这个问题啦！如此说来， script 确实是需要学习的啊！

时至今日，虽然 /etc/init.d/* 这个脚本启动的方式 （systemV） 已经被新一代的 systemd 所取代 （从 CentOS 7 开始）， 但是很多的个别服务在管理他们的服务启动方面，还是使用 shell script 的机制喔！所以，最好还是能够熟悉啦！

- 简单入侵侦测功能

当我们的系统有异状时，大多会将这些异状记录在系统记录器，也就是我们常提到的"系统登录文件"， 那么我们可以在固定的几分钟内主动的去分析系统登录文件，若察觉有问题，就立刻通报管理员， 或者是立刻加强防火墙的设置规则，如此一来，你的主机可就能够达到"自我保护"的聪明学习功能啦～ 举例来说，我们可以通过 shell script 去分析"当该封包尝试几次还是连线失败之后，就予以抵挡住该 IP"之类的举动，例如鸟哥写过一个关于抵挡砍站软件的 shell script ， 就是用这个想法去达成的呢！

- 连续指令单一化

其实，对于新手而言， script 最简单的功能就是："汇整一些在 command line 下达的连续指令，将他写入 scripts 当中，而由直接执行 scripts 来启动一连串的 command line 指令输入！"例如： 防火墙连续规则 （iptables） ，开机载入程序的项目 （就是在 /etc/rc.d/rc.local 里头的数据） ，等等都是相似的功能啦！ 其实，说穿了，如果不考虑 program 的部分，那么 scripts 也可以想成"仅是帮我们把一大串的指令汇整在一个文件里面， 而直接执行该文件就可以执行那一串又臭又长的指令段！"就是这么简单啦！

- 简易的数据处理

由前一章正则表达式的 awk 程序说明中， 你可以发现， awk 可以用来处理简单的数据数据呢！例如薪资单的处理啊等等的。 shell script 的功能更强大，例如鸟哥曾经用 shell script 直接处理数据数据的比对啊， 文字数据的处理啊等等的，撰写方便，速度又快（因为在 Linux 性能较佳） ，真的是很不错用的啦！

举例来说，鸟哥每学期都得要以学生的学号来创建他们能够操作 Linux 的系统帐号，然后每个帐号还得要能够有磁盘容量的限制 （quota） 以及相关的设置等等， 那因为学校的校务系统提供的数据都是一整串学生信息，并没有单纯的学号字段，所以鸟哥就得要通过前几章的

方法搭配 shell script 来自动处理相关设置流程， 这样才不会每学期都头疼一次啊！

- 跨平台支持与学习历程较短

几乎所有的 Unix Like 上面都可以跑 shell script ，连 MS Windows 系列也有相关的 script 仿真器可以用， 此外， shell script 的语法是相当友好的，看都看的懂得文字 （虽然是英文），而不是机器码， 很容易学习～这些都是你可以加以考虑的学习点啊！

上面这些都是你考虑学习 shell script 的特点～此外， shell script 还可以简单的以 vim 来直接编写，实在是很方便的好东西！所以，还是建议你学习一下啦。

不过，虽然 shell script 号称是程序 （program） ，但实际上， shell script 处理数据的速度上是不太够的。因为 shell script 用的是外部的指令与 bash shell 的一些默认工具，所以，他常常会去调用外部的函数库，因此，运算速度上面当然比不上传统的程序语言。 所以啰， shell script 用在系统管理上面是很好的一项工具，但是用在处理大量数值运算上， 就不够好了，因为 Shell scripts 的速度较慢，且使用的 CPU 资源较多，造成主机资源的分配不良。还好，我们通常利用 shell script 来处理服务器的侦测，倒是没有进行大量运算的需求啊！所以不必担心的啦！

12.1.2 第一支 script 的撰写与执行

如同前面讲到的，shell script 其实就是纯文本文件，我们可以编辑这个文件，然后让这个文件来帮我们一次执行多个指令， 或者是利用一些运算与逻辑判断来帮我们达成某些功能。所以啦，要编辑这个文件的内容时，当然就需要具备有 bash 指令下达的相关认识。下达指令需要注意的事项在第四章的开始下达指令小节内已经提过，有疑问请自行回去翻阅。 在 shell script 的撰写中还需要用到下面的注意事项：

1. 指令的执行是从上而下、从左而右的分析与执行；
2. 指令的下达就如同第四章内提到的： 指令、选项与参数间的多个空白都会被忽略掉；
3. 空白行也将被忽略掉，并且 [tab] 按键所推开的空白同样视为空白键；
4. 如果读取到一个 Enter 符号 （CR） ，就尝试开始执行该行 （或该串） 命令；
5. 至于如果一行的内容太多，则可以使用“ [Enter] ”来延伸至下一行；
6. “ # ”可做为注解！任何加在 # 后面的数据将全部被视为注解文字而被忽略！

如此一来，我们在 script 内所撰写的程序，就会被一行一行的执行。现在我们假设你写的这个程序文件名是 /home/dmtsai/shell.sh 好了，那如何执行这个文件？很简单，可以有下面几个方法：

- 直接指令下达： shell.sh 文件必须要具备可读与可执行 （rx） 的权限，然后：
    - 绝对路径：使用 /home/dmtsai/shell.sh 来下达指令；
    - 相对路径：假设工作目录在 /home/dmtsai/ ，则使用 ./shell.sh 来执行
    - 变量“PATH”功能：将 shell.sh 放在 PATH 指定的目录内，例如： ~/bin/
- 以 bash 程序来执行：通过“ bash shell.sh ”或“ sh shell.sh ”来执行

反正重点就是要让那个 shell.sh 内的指令可以被执行的意思啦！ 咦！那我为何需要使用“./shell.sh ”来下达指令？忘记了吗？回去第十章内的指令搜寻顺序察看一下， 你就会知道原因了！同时，由于 CentOS 默认使用者主文件夹下的 ~/bin 目录会被设置到 ${PATH} 内，所以你也可以将 shell.sh 创建在 /home/dmtsai/bin/ 下面 （ ~/bin 目录需要自行设置） 。此时，若 shell.sh 在 ~/bin 内且具有 rx 的权限，那就直接输入 shell.sh 即可执行该脚本程序！

那为何“ sh shell.sh ”也可以执行呢？这是因为 /bin/sh 其实就是 /bin/bash （链接文件），使用 sh shell.sh 亦即告诉系统，我想要直接以 bash 的功能来执行 shell.sh 这个文件内的相关指令的意思，所以此时你的 shell.sh 只要有 r 的权限即可被执行喔！而我们也可以利用 sh 的参数，如 -n 及 -x 来检查与追踪 shell.sh 的语法是否正确呢！ ^_^

- 撰写第一支 script

在武侠世界中，不论是那个门派，要学武功要从扫地与蹲马步做起，那么要学程序呢？呵呵，肯定是由“秀出 Hello World！” 这个字眼开始的！OK！那么鸟哥就先写一支 script 给大家瞧一瞧：

```
[dmtsai@study ~]$ mkdir bin; cd bin
[dmtsai@study bin]$ vim hello.sh
#!/bin/bash
# Program:
#       This program shows "Hello World!" in your screen.
# History:
# 2015/07/16    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
echo -e "Hello World! \a \n"
exit 0
```

在本章当中，请将所有撰写的 script 放置到你主文件夹的 ~/bin 这个目录内，未来比较好管理啦！上面的写法当中，鸟哥主要将整个程序的撰写分成数段，大致是这样：

1. 第一行 #!/bin/bash 在宣告这个 script 使用的 shell 名称： 因为我们使用的是 bash ，所以，必须要以“ **#!/bin/bash** ”来宣告这个文件内的语法使用 bash 的语法！那么当这个程序被执行时，他就能够载入 bash 的相关环境配置文件 （一般来说就是 non-login shell 的 ~/.bashrc）， 并且执行 bash 来使我们下面的指令能够执行！这很重要的！（在很多状况中，如果没有设置好这一行， 那么该程序很可能会无法执行，因为系统可能无法判断该程序需要使用什么 shell 来执行啊！）

2. 程序内容的说明： 整个 script 当中，除了第一行的“ #! ”是用来宣告 shell 的之外，其他的 # 都是“注解”用途！ 所以上面的程序当中，第二行以下就是用来说明整个程序的基本数据。一般来说， 建议你一定要养成说明该 script 的：1. 内容与功能； 2. 版本信息； 3. 作者与联络方式； 4. 创建日期；5. 历史纪录 等等。这将有助于未来程序的改写与 debug 呢！

3. 主要环境变量的宣告： 建议务必要将一些重要的环境变量设置好，鸟哥个人认为， PATH 与 LANG （如果有使用到输出相关的信息时） 是当中最重要的！ 如此一来，则可让我们这支程序在进行时，可以直接下达一些外部指令，而不必写绝对路径呢！比较方

便啦！

4. 主要程序部分 就将主要的程序写好即可！在这个例子当中，就是 echo 那一行啦！
5. 执行成果告知 （定义回传值） 是否记得我们在第十章里面要讨论一个指令的执行成功与否，可以使用 $? 这个变量来观察～ 那么我们也可以利用 exit 这个指令来让程序中断，并且回传一个数值给系统。 在我们这个例子当中，鸟哥使用 exit 0 ，这代表离开 script 并且回传一个 0 给系统， 所以我执行完这个 script 后，若接着下达 echo $? 则可得到 0 的值喔！ 更聪明的读者应该也知道了，呵呵！利用这个 exit n （n 是数字） 的功能，我们还可以自订错误讯息， 让这支程序变得更加的 smart 呢！

接下来通过刚刚上头介绍的执行方法来执行看看结果吧！

```
[dmtsai@study bin]$ sh hello.sh
Hello World !
```

你会看到屏幕是这样，而且应该还会听到“咚”的一声，为什么呢？还记得前一章提到的 printf 吧？用 echo 接着那些特殊的按键也可以发生同样的事情～ 不过， echo 必须要加上 -e 的选项才行！呵呵！在你写完这个小 script 之后，你就可以大声的说：“我也会写程序了”！哈哈！很简单有趣吧～ ^_^

另外，你也可以利用：“chmod a+x hello.sh; ./hello.sh”来执行这个 script 的呢！

12.1.3 撰写 shell script 的良好习惯创建

一个良好习惯的养成是很重要的～大家在刚开始撰写程序的时候，最容易忽略这部分， 认为程序写出来就好了，其他的不重要。其实，如果程序的说明能够更清楚，那么对你自己是有很大的帮助的。

举例来说，鸟哥自己为了自己的需求，曾经撰写了不少的 script 来帮我进行主机 IP 的侦测啊、 登录文件分析与管理啊、自动上传下载重要配置文件啊等等的，不过，早期就是因为太懒了， 管理的主机又太多了，常常同一个程序在不同的主机上面进行更改，到最后，到底哪一支才是最新的都记不起来， 而且，重点是，我到底是改了哪里？为什么做那样的修改？都忘的一干二净～真要命～

所以，后来鸟哥在写程序的时候，通常会比较仔细的将程序的设计过程给他记录下来，而且还会记录一些历史纪录， 如此一来，好多了～至少很容易知道我修改了哪些数据，以及程序修改的理念与逻辑概念等等， 在维护上面是轻松很多很多的喔！

另外，在一些环境的设置上面，毕竟每个人的环境都不相同，为了取得较佳的执行环境， 我都会自行先定义好一些一定会被用到的环境变量，例如 PATH 这个玩意儿！ 这样比较好啦～所以说，建议你一定要养成良好的 script 撰写习惯，在每个 script 的文件开始处记录好：

- script 的功能；
- script 的版本信息；

- script 的作者与联络方式；
- script 的版权宣告方式；
- script 的 History （历史纪录）；
- script 内较特殊的指令，使用“绝对路径”的方式来下达；
- script 运行时需要的环境变量预先宣告与设置。

除了记录这些信息之外，在较为特殊的程序码部分，个人建议务必要加上注解说明，可以帮助你非常非常多！ 此外，程序码的撰写最好使用巢状方式，在包覆的内部程序码最好能以 [tab] 按键的空格向后推， 这样你的程序码会显的非常的漂亮与有条理！在查阅与 debug 上较为轻松愉快喔！ 另外，使用撰写 script 的工具最好使用 vim 而不是 vi ，因为 vim 会有额外的语法检验机制，能够在第一阶段撰写时就发现语法方面的问题喔！

12.2 简单的 shell script 练习

在第一支 shell script 撰写完毕之后，相信你应该具有基本的撰写功力了。接下来，在开始更深入的程序概念之前，我们先来玩一些简单的小范例好了。 下面的范例中，达成结果的方式相当的多，建议你先自行撰写看看，写完之后再与鸟哥写的内容比对， 这样才能更加深概念喔！好！不啰唆，我们就一个一个来玩吧！

12.2.1 简单范例

下面的范例在很多的脚本程序中都会用到，而下面的范例又都很简单！值得参考看看喔！

- 对谈式脚本：变量内容由使用者决定

很多时候我们需要使用者输入一些内容，好让程序可以顺利运行。 简单的来说，大家应该都有安装过软件的经验，安装的时候，他不是会问你“要安装到那个目录去”吗？ 那个让使用者输入数据的动作，就是让使用者输入变量内容啦。

你应该还记得在十章 bash 的时候，我们有学到一个 read 指令吧？现在，请你以 read 指令的用途，撰写一个 script ，他可以让使用者输入：1. first name 与 2. last name， 最后并且在屏幕上显示：“Your full name is: ”的内容：

```
[dmtsai@study bin]$ vim showname.sh
#!/bin/bash
# Program:
#    User inputs his first name and last name.  Program shows his full name.
# History:
# 2015/07/16    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

read -p "Please input your first name: " firstname      # 提示使用者输入
read -p "Please input your last name:  " lastname       # 提示使用者输入
echo -e "\nYour full name is: ${firstname} ${lastname}" # 结果由屏幕输出
```

将上面这个 showname.sh 执行一下，你就能够发现使用者自己输入的变量可以让程序所取用，并且将他显示到屏幕上！ 接下来，如果想要制作一个每次执行都会依据不同的日期而变化结果的脚本呢？

- 随日期变化：利用 date 进行文件的创建

想像一个状况，假设我的服务器内有数据库，数据库每天的数据都不太一样，因此当我备份时，希望将每天的数据都备份成不同的文件名， 这样才能够让旧的数据也能够保存下来不被覆盖。哇！不同文件名呢！这真困扰啊？难道要我每天去修改 script ？

不需要啊！考虑每天的“日期”并不相同，所以我可以将文件名取成类似： backup.2015-07-16.data ， 不就可以每天一个不同文件名了吗？呵呵！确实如此。那个 2015-07-16 怎么来的？那就是重点啦！接下来出个相关的例子： 假设我想要创建三个空的文件 （通过 touch）

，文件名最开头由使用者输入决定，假设使用者输入 filename 好了，那今天的日期是 2015/07/16 ， 我想要以前天、昨天、今天的日期来创建这些文件，亦即 filename_20150714, filename_20150715, filename_20150716 ，该如何是好？

```
[dmtsai@study bin]$ vim create_3_filename.sh
#!/bin/bash
# Program:
#    Program creates three files, which named by user's input and date command.
# History:
# 2015/07/16    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

# 1\. 让使用者输入文件名称，并取得 fileuser 这个变量；
echo -e "I will use 'touch' command to create 3 files." # 纯粹显示信息
read -p "Please input your filename: " fileuser         # 提示使用者输入

# 2\. 为了避免使用者随意按 Enter ，利用[变量功能](../Text/index.html#variable_other_re)分析文件名
filename=${fileuser:-"filename"}           # 开始判断有否配置文件名

# 3\. 开始利用 date 指令来取得所需要的文件名了；
date1=$（date --date='2 days ago' +%Y%m%d）  # 前两天的日期
date2=$（date --date='1 days ago' +%Y%m%d）  # 前一天的日期
date3=$（date +%Y%m%d）                      # 今天的日期
file1=${filename}${date1}                  # 下面三行在配置文件名
file2=${filename}${date2}
file3=${filename}${date3}

# 4\. 将文件名创建吧！
touch "${file1}"                           # 下面三行在创建文件
touch "${file2}"
touch "${file3}"
```

上面的范例鸟哥使用了很多在第十章介绍过的概念： 包括小指令“ $（command） ”的取得讯息、变量的设置功能、变量的累加以及利用 touch 指令辅助！ 如果你开始执行这个 create_3_filename.sh 之后，你可以进行两次执行：一次直接按 [Enter] 来查阅文件名是啥？ 一次可以输入一些字符，这样可以判断你的脚本是否设计正确喔！

- 数值运算：简单的加减乘除

各位看官应该还记得，我们可以使用 declare 来定义变量的类型吧？ 当变量定义成为整数后才能够进行加减运算啊！此外，我们也可以利用“ $（（计算式）） ”来进行数值运算的。 可惜的是， bash shell 里头默认仅支持到整数的数据而已。OK！那我们来玩玩看，如果我们要使用者输入两个变量， 然后将两个变量的内容相乘，最后输出相乘的结果，那可以怎么做？

```
[dmtsai@study bin]$ vim multiplying.sh
#!/bin/bash
# Program:
#    User inputs 2 integer numbers; program will cross these two numbers.
# History:
# 2015/07/16    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
echo -e "You SHOULD input 2 numbers, I will multiplying them! \n"
read -p "first number:  " firstnu
read -p "second number: " secnu
total=$（（${firstnu}*${secnu}））
echo -e "\nThe result of ${firstnu} x ${secnu} is ==&gt; ${total}"
```

在数值的运算上，我们可以使用“ declare -i total=${firstnu}*${secnu} ” 也可以使用上面的方式来进行！基本上，鸟哥比较建议使用这样的方式来进行运算：

> var=$（（运算内容））

不但容易记忆，而且也比较方便的多，因为两个小括号内可以加上空白字符喔！ 未来你可以使用这种方式来计算的呀！至于数值运算上的处理，则有：“ +, -, , /, % ”等等。 那个 % 是取余数啦～举例来说， *13* 对 *3* 取余数，结果是 *13=43+1*，所以余数是 1 啊！就是：

```
[dmtsai@study bin]$ echo $（（ 13 % 3 ））
1
```

这样了解了吧？另外，如果你想要计算含有小数点的数据时，其实可以通过 bc 这个指令的协助喔！ 例如可以这样做：

```
[dmtsai@study bin]$ echo "123.123*55.9" &#124; bc
6882.575
```

了解了 bc 的妙用之后，来让我们测试一下如何计算 pi 这个东西呢？

- 数值运算：通过 bc 计算 pi

其实计算 pi 时，小数点以下位数可以无限制的延伸下去！而 bc 有提供一个运算 pi 的函数，只是想要使用该函数必须要使用 bc -l 来调用才行。 也因为这个小数点以下位数可以无线延伸运算的特性存在，所以我们可以通过下面这只小脚本来让使用者输入一个“小数点为数值”，以让 pi 能够更准确！

```
[dmtsai@study bin]$ vim cal_pi.sh
#!/bin/bash
# Program:
#     User input a scale number to calculate pi number.
# History:
# 2015/07/16     VBird     First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
echo -e "This program will calculate pi value. \n"
echo -e "You should input a float number to calculate pi value.\n"
read -p "The scale number （10~10000） ? " checking
num=${checking:-"10"}           # 开始判断有否有输入数值
echo -e "Starting calcuate pi value.  Be patient."
time echo "scale=${num}; 4*a（1）" &#124; bc -lq
```

上述数据中，那个 4*a（1） 是 bc 主动提供的一个计算 pi 的函数，至于 scale 就是要 bc 计算几个小数点下位数的意思。当 scale 的数值越大， 代表 pi 要被计算的越精确，当然用掉的时间就会越多！因此，你可以尝试输入不同的数值看看！不过，最好是不要超过 5000 啦！因为会算很久！ 如果你要让你的 CPU 随时保持在高负载，这个程序算下去你就会知道有多操 CPU 啰！ ^_^



Tips 鸟哥的实验室中，为了要确认虚拟机的效率问题，所以很多时候需要保持虚拟机在忙碌的状态～鸟哥的学生就是丢这只程序进去系统跑！ 但是将 scale 调高一些，那计算就得要花比较多时间～用以达到我们需要 CPU 忙碌的状态喔！

12.2.2 script 的执行方式差异 （source, sh script, ./script）

不同的 script 执行方式会造成不一样的结果喔！尤其影响 bash 的环境很大呢！脚本的执行方式除了前面小节谈到的方式之外，还可以利用 source 或小数点 （.） 来执行喔！那么这种执行方式有何不同呢？当然是不同的啦！让我们来说说！

- 利用直接执行的方式来执行 script

当使用前一小节提到的直接指令下达 （不论是绝对路径/相对路径还是 ${PATH} 内），或者是利用 bash （或 sh） 来下达脚本时， 该 script 都会使用一个新的 bash 环境来执行脚本内的指令！也就是说，使用这种执行方式时， 其实 script 是在子程序的 bash 内执行的！我们在第十章 BASH 内谈到 export 的功能时，曾经就父程序/子程序谈过一些概念性的问题， 重点在于："当子程序完成后，在子程序内的各项变量或动作将会结束而不会传回到父程序中"！ 这是什么意思呢？

我们举刚刚提到过的 showname.sh 这个脚本来说明好了，这个脚本可以让使用者自行设置两个变量，分别是 firstname 与 lastname，想一想，如果你直接执行该指令时，该指令帮你设置的 firstname 会不会生效？看一下下面的执行结果：

```
[dmtsai@study bin]$ echo ${firstname} ${lastname}
    &lt;==确认了，这两个变量并不存在喔！
[dmtsai@study bin]$ sh showname.sh
Please input your first name: VBird &lt;==这个名字是鸟哥自己输入的
Please input your last name:  Tsai

Your full name is: VBird Tsai      &lt;==看吧！在 script 运行中，这两个变量有生效
[dmtsai@study bin]$ echo ${firstname} ${lastname}
    &lt;==事实上，这两个变量在父程序的 bash 中还是不存在的！
```

上面的结果你应该会觉得很奇怪，怎么我已经利用 showname.sh 设置好的变量竟然在 bash 环境下面无效！怎么回事呢？ 如果将程序相关性绘制成图的话，我们以下图来说明。当你使用直接执行的方法来处理时，系统会给予一支新的 bash 让我们来执行 showname.sh 里面的指令，因此你的 firstname, lastname 等变量其实是在下图中的子程序 bash 内执行的。 当 showname.sh 执行完毕后，子程序 bash 内的所有数据便被移除，因此上表的练习中，在父程序下面 echo ${firstname} 时， 就看不到任何东西了！这样可以理解吗？

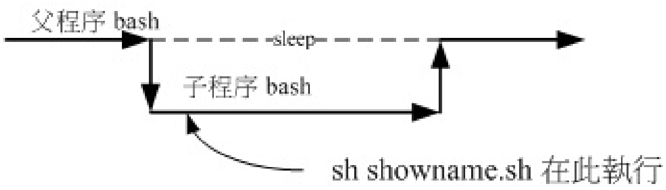


图12.2.1、showname.sh 在子程序当中运行的示意图

- 利用 source 来执行脚本：在父程序中执行

如果你使用 source 来执行指令那就不一样了！同样的脚本我们来执行看看：

```
[dmtsai@study bin]$ source showname.sh
Please input your first name: VBird
Please input your last name:  Tsai

Your full name is: VBird Tsai
[dmtsai@study bin]$ echo ${firstname} ${lastname}
VBird Tsai  &lt;==嘿嘿！有数据产生喔！
```

竟然生效了！没错啊！因为 source 对 script 的执行方式可以使用下面的图示来说明！showname.sh 会在父程序中执行的，因此各项动作都会在原本的 bash 内生效！这也是为啥你不登出系统而要让某些写入 ~/.bashrc 的设置生效时，需要使用“ source ~/.bashrc ”而不能使用“ bash ~/.bashrc ”是一样的啊！

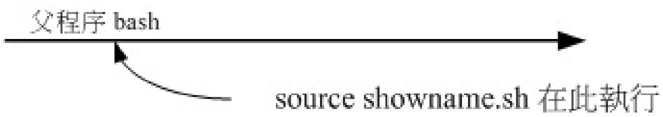


图12.2.2、showname.sh 在父程序当中运行的示意图

12.3 善用判断式

在第十章中，我们提到过 $? 这个变量所代表的意义， 此外，也通过 && 及 || 来作为前一个指令执行回传值对于后一个指令是否要进行的依据。第十章的讨论中，如果想要判断一个目录是否存在， 当时我们使用的是 ls 这个指令搭配数据流重导向，最后配合 $? 来决定后续的指令进行与否。 但是否有更简单的方式可以来进行“条件判断”呢？有的～那就是“ test ”这个指令。

12.3.1 利用 test 指令的测试功能

当我要检测系统上面某些文件或者是相关的属性时，利用 test 这个指令来工作真是好用得不得了， 举例来说，我要检查 /dmtsai 是否存在时，使用：

```
[dmtsai@study ~]$ test -e /dmtsai
```

执行结果并不会显示任何讯息，但最后我们可以通过 $? 或 && 及 || 来展现整个结果呢！ 例如我们在将上面的例子改写成这样：

```
[dmtsai@study ~]$ test -e /dmtsai && echo "exist" &#124;&#124; echo "Not exist"
Not exist  &lt;==结果显示不存在啊！
```

最终的结果可以告知我们是“exist”还是“Not exist”呢！那我知道 -e 是测试一个“东西”在不在，如果还想要测试一下该文件名是啥玩意儿时，还有哪些标志可以来判断的呢？呵呵！有下面这些东西喔！

测试的标志	代表意义
1. 关于某个文件名的“文件类型”判断，如 test -e filename 表示存在否	
-e	该“文件名”是否存在？（常用）
-f	该“文件名”是否存在且为文件（file）？（常用）
-d	该“文件名”是否存在且为目录（directory）？（常用）
-b	该“文件名”是否存在且为一个 block device 设备？
-c	该“文件名”是否存在且为一个 character device 设备？
-S	该“文件名”是否存在且为一个 Socket 文件？
-p	该“文件名”是否存在且为一个 FIFO （pipe） 文件？

-L	该“文件名”是否存在且为一个链接文件？
2. 关于文件的权限侦测，如 test -r filename 表示可读否 （但 root 权限常有例外）	
-r	侦测该文件名是否存在且具有“可读”的权限？
-w	侦测该文件名是否存在且具有“可写”的权限？
-x	侦测该文件名是否存在且具有“可执行”的权限？
-u	侦测该文件名是否存在且具有“SUID”的属性？
-g	侦测该文件名是否存在且具有“SGID”的属性？
-k	侦测该文件名是否存在且具有“Sticky bit”的属性？
-s	侦测该文件名是否存在且为“非空白文件”？
3. 两个文件之间的比较，如： test file1 -nt file2	
-nt	（newer than）判断 file1 是否比 file2 新
-ot	（older than）判断 file1 是否比 file2 旧
-ef	判断 file1 与 file2 是否为同一文件，可用在判断 hard link 的判定上。 主要意义在判定，两个文件是否均指向同一个 inode 哩！
4. 关于两个整数之间的判定，例如 test n1 -eq n2	
-eq	两数值相等 （equal）
-ne	两数值不等 （not equal）
-gt	n1 大于 n2 （greater than）
-lt	n1 小于 n2 （less than）
-ge	n1 大于等于 n2 （greater than or equal）
-le	n1 小于等于 n2 （less than or equal）
5. 判定字串的数据	
test -z string	判定字串是否为 0 ？若 string 为空字串，则为 true
test -n string	判定字串是否非为 0 ？若 string 为空字串，则为 false。 -n 亦可省略
test str1 == str2	判定 str1 是否等于 str2 ，若相等，则回传 true
test str1 != str2	判定 str1 是否不等于 str2 ，若相等，则回传 false
6. 多重条件判定，例如： test -r filename -a -x filename	
	（and）两状况同时成立！例如 test -r file -a -x file，

	则 file 同时具有 r 与 x 权限时，才回传 true。
-o	（or）两状况任何一个成立！例如 test -r file -o -x file，则 file 具有 r 或 x 权限时，就可回传 true。
!	反相状态，如 test ! -x file ，当 file 不具有 x 时，回传 true

OK！现在我们就利用 test 来帮我们写几个简单的例子。首先，判断一下，让使用者输入一个文件名，我们判断：

1. 这个文件是否存在，若不存在则给予一个“Filename does not exist”的讯息，并中断程序；
2. 若这个文件存在，则判断他是个文件或目录，结果输出“Filename is regular file”或“Filename is directory”
3. 判断一下，执行者的身份对这个文件或目录所拥有的权限，并输出权限数据！

你可以先自行创作看看，然后再跟下面的结果讨论讨论。注意利用 test 与 && 还有 || 等标志！

```
[dmtsai@study bin]$ vim file_perm.sh
#!/bin/bash
# Program:
#    User input a filename, program will check the flowing:
#    1.） exist? 2.） file/directory? 3.） file permissions
# History:
# 2015/07/16    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

# 1\. 让使用者输入文件名，并且判断使用者是否真的有输入字串？
echo -e "Please input a filename, I will check the filename's type and permission. \n\n"
read -p "Input a filename : " filename
test -z ${filename} && echo "You MUST input a filename." && exit 0
# 2\. 判断文件是否存在？若不存在则显示讯息并结束脚本
test ! -e ${filename} && echo "The filename '${filename}' DO NOT exist" && exit 0
# 3\. 开始判断文件类型与属性
test -f ${filename} && filetype="regulare file"
test -d ${filename} && filetype="directory"
test -r ${filename} && perm="readable"
test -w ${filename} && perm="${perm} writable"
test -x ${filename} && perm="${perm} executable"
# 4\. 开始输出信息！
echo "The filename: ${filename} is a ${filetype}"
echo "And the permissions for you are : ${perm}"
```

如果你执行这个脚本后，他会依据你输入的文件名来进行检查喔！先看是否存在，再看为文件或目录类型，最后判断权限。 但是你必须要注意的是，由于 root 在很多权限的限制上面都是无效的，所以使用 root 执行这个脚本时， 常常会发现与 ls -l 观察到的结果并不相同！所以，建议使用一般使用者来执行这个脚本试看看。

12.3.2 利用判断符号 [ ]

除了我们很喜欢使用的 test 之外，其实，我们还可以利用判断符号“ [ ] ”（就是中括号啦） 来进行数据的判断呢！ 举例来说，如果我想要知道 ${HOME} 这个变量是否为空的，可以这样做：

```
[dmtsai@study ~]$ [ -z "${HOME}" ] ; echo $?
```

使用中括号必须要特别注意，因为中括号用在很多地方，包括万用字符与正则表达式等等，所以如果要在 bash 的语法当中使用中括号作为 shell 的判断式时，必须要注意中括号的两端需要有空白字符来分隔喔！ 假设我空白键使用“□”符号来表示，那么，在这些地方你都需要有空白键：

```
[  "$HOME"  ==  "$MAIL"  ]
[□"$HOME"□==□"$MAIL"□]
 ↑       ↑  ↑       ↑
```



Tips 你会发现鸟哥在上面的判断式当中使用了两个等号“ == ”。其实在 bash 当中使用一个等号与两个等号的结果是一样的！ 不过在一般惯用程序的写法中，一个等号代表“变量的设置”，两个等号则是代表“逻辑判断 （是与否之意）”。 由于我们在中括号内重点在于“判断”而非“设置变量”，因此鸟哥建议您还是使用两个等号较佳！

上面的例子在说明，两个字串 ${HOME} 与 ${MAIL} 是否相同的意思，相当于 test ${HOME} == ${MAIL} 的意思啦！ 而如果没有空白分隔，例如 [${HOME}==${MAIL}] 时，我们的 bash 就会显示错误讯息了！这可要很注意啊！ 所以说，你最好要注意：

- 在中括号 [] 内的每个元件都需要有空白键来分隔；
- 在中括号内的变量，最好都以双引号括号起来；
- 在中括号内的常数，最好都以单或双引号括号起来。

为什么要这么麻烦啊？直接举例来说，假如我设置了 name="VBird Tsai" ，然后这样判定：

```
[dmtsai@study ~]$ name="VBird Tsai"
[dmtsai@study ~]$ [ ${name} == "VBird" ]
bash: [: too many arguments
```

见鬼了！怎么会发生错误啊？bash 还跟我说错误是由于“太多参数 （arguments）”所致！ 为什么呢？因为 ${name} 如果没有使用双引号刮起来，那么上面的判定式会变成：

> [VBird Tsai == "VBird"]

上面肯定不对嘛！因为一个判断式仅能有两个数据的比对，上面 VBird 与 Tsai 还有 "VBird" 就有三个数据！ 这不是我们要的！我们要的应该是下面这个样子：

> ["VBird Tsai" == "VBird"]

这可是差很多的喔！另外，中括号的使用方法与 test 几乎一模一样啊～ 只是中括号比较常用在条件判断式 if ..... then ..... fi 的情况中就是了。 好，那我们也使用中括号的判断来做一个小案例好了，案例设置如下：

1. 当执行一个程序的时候，这个程序会让使用者选择 Y 或 N ，
2. 如果使用者输入 Y 或 y 时，就显示“ OK, continue ”
3. 如果使用者输入 n 或 N 时，就显示“ Oh, interrupt ！”
4. 如果不是 Y/y/N/n 之内的其他字符，就显示“ I don't know what your choice is ”

利用中括号、 && 与 || 来继续吧！

```
[dmtsai@study bin]$ vim ans_yn.sh
#!/bin/bash
# Program:
#     This program shows the user's choice
# History:
# 2015/07/16    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

read -p "Please input （Y/N）: " yn
[ "${yn}" == "Y" -o "${yn}" == "y" ] && echo "OK, continue" && exit 0
[ "${yn}" == "N" -o "${yn}" == "n" ] && echo "Oh, interrupt!" && exit 0
echo "I don't know what your choice is" && exit 0
```

由于输入正确 （Yes） 的方法有大小写之分，不论输入大写 Y 或小写 y 都是可以的，此时判断式内就得要有两个判断才行！ 由于是任何一个成立即可 （大写或小写的 y） ，所以这里使用 -o （或） 链接两个判断喔！ 很有趣吧！利用这个字串判别的方法，我们就可以很轻松的将使用者想要进行的工作分门别类呢！ 接下来，我们再来谈一些其他有的没有的东西吧！

12.3.3 Shell script 的默认变量（$0, $1...）

我们知道指令可以带有选项与参数，例如 ls -la 可以察看包含隐藏文件的所有属性与权限。那么 shell script 能不能在脚本文件名后面带有参数呢？很有趣喔！举例来说，如果你想要重新启动系统的网络，可以这样做：

```
[dmtsai@study ~]$ file /etc/init.d/network
/etc/init.d/network: Bourne-Again shell script, ASCII text executable
# 使用 file 来查询后，系统告知这个文件是个 bash 的可执行 script 喔！
[dmtsai@study ~]$ /etc/init.d/network restart
```

restart 是重新启动的意思，上面的指令可以“重新启动 /etc/init.d/network 这支程序”的意思！唔！那么如果你在 /etc/init.d/network 后面加上 stop 呢？没错！就可以直接关闭该服务了！这么神奇啊？ 没错啊！如果你要依据程序的执行给予一些变量去进行不同的任务时，本章一开

始是使用 read 的功能！但 read 功能的问题是你得要手动由键盘输入一些判断式。如果通过指令后面接参数， 那么一个指令就能够处理完毕而不需要手动再次输入一些变量行为！这样下达指令会比较简单方便啦！

script 是怎么达成这个功能的呢？其实 script 针对参数已经有设置好一些变量名称了！对应如下：

```
/path/to/scriptname  opt1  opt2  opt3  opt4
       $0             $1    $2    $3    $4
```

这样够清楚了吧？执行的脚本文件名为 $0 这个变量，第一个接的参数就是 $1 啊～ 所以，只要我们在 script 里面善用 $1 的话，就可以很简单的立即下达某些指令功能了！除了这些数字的变量之外， 我们还有一些较为特殊的变量可以在 script 内使用来调用这些参数喔！

- $# ：代表后接的参数“个数”，以上表为例这里显示为“ 4 ”；
- $@ ：代表“ "$1" "$2" "$3" "$4" ”之意，每个变量是独立的（用双引号括起来）；
- $* ：代表“ "$1<u>c</u>$2<u>c</u>$3<u>c</u>$4" ”，其中 <u>c</u> 为分隔字符，默认为空白键， 所以本例中代表“ "$1 $2 $3 $4" ”之意。

那个 $@ 与 $* 基本上还是有所不同啦！不过，一般使用情况下可以直接记忆 $@ 即可！ 好了，来做个例子吧～假设我要执行一个可以携带参数的 script ，执行该脚本后屏幕会显示如下的数据：

- 程序的文件名为何？
- 共有几个参数？
- 若参数的个数小于 2 则告知使用者参数数量太少
- 全部的参数内容为何？
- 第一个参数为何？
- 第二个参数为何

```
[dmtsai@study bin]$ vim how_paras.sh
#!/bin/bash
# Program:
#    Program shows the script name, parameters...
# History:
# 2015/07/16    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

echo "The script name is        ==&gt; ${0}"
echo "Total parameter number is ==&gt; $#"
[ "$#" -lt 2 ] && echo "The number of parameter is less than 2\.  Stop here." && exit 0
echo "Your whole parameter is   ==&gt; '$@'"
echo "The 1st parameter         ==&gt; ${1}"
echo "The 2nd parameter         ==&gt; ${2}"
```

执行结果如下：

```
[dmtsai@study bin]$ sh how_paras.sh theone haha quot
The script name is        ==&gt; how_paras.sh       &lt;==文件名
Total parameter number is ==&gt; 3                  &lt;==果然有三个参数
Your whole parameter is   ==&gt; 'theone haha quot' &lt;==参数的内容全部
The 1st parameter         ==&gt; theone             &lt;==第一个参数
The 2nd parameter         ==&gt; haha               &lt;==第二个参数
```

- shift：造成参数变量号码偏移

除此之外，脚本后面所接的变量是否能够进行偏移 （shift） 呢？什么是偏移啊？我们直接以下面的范例来说明好了， 用范例说明比较好解释！我们将 how_paras.sh 的内容稍作变化一下，用来显示每次偏移后参数的变化情况：

```
[dmtsai@study bin]$ vim shift_paras.sh
#!/bin/bash
# Program:
#    Program shows the effect of shift function.
# History:
# 2009/02/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

echo "Total parameter number is ==&gt; $#"
echo "Your whole parameter is   ==&gt; '$@'"
shift   # 进行第一次“一个变量的 shift ”
echo "Total parameter number is ==&gt; $#"
echo "Your whole parameter is   ==&gt; '$@'"
shift 3 # 进行第二次“三个变量的 shift ”
echo "Total parameter number is ==&gt; $#"
echo "Your whole parameter is   ==&gt; '$@'"
```

这玩意的执行成果如下：

```
[dmtsai@study bin]$ sh shift_paras.sh one two three four five six &lt;==给予六个参数
Total parameter number is ==&gt; 6   &lt;==最原始的参数变量情况
Your whole parameter is   ==&gt; 'one two three four five six'
Total parameter number is ==&gt; 5   &lt;==第一次偏移，看下面发现第一个 one 不见了
Your whole parameter is   ==&gt; 'two three four five six'
Total parameter number is ==&gt; 2   &lt;==第二次偏移掉三个，two three four 不见了
Your whole parameter is   ==&gt; 'five six'
```

光看结果你就可以知道啦，那个 shift 会移动变量，而且 shift 后面可以接数字，代表拿掉最前面的几个参数的意思。 上面的执行结果中，第一次进行 shift 后他的显示情况是“ one two three four five six”，所以就剩下五个啦！第二次直接拿掉三个，就变成“ two three four five six ”啦！ 这样这个案例可以了解了吗？理解了 shift 的功能了吗？

上面这几个例子都很简单吧？几乎都是利用 bash 的相关功能而已～ 不难啦～下面我们就要使用条件判断式来进行一些分别功能的设置了，好好瞧一瞧先～

12.4 条件判断式

只要讲到“程序”的话，那么条件判断式，亦即是“ if then ”这种判别式肯定一定要学习的！ 因为很多时候，我们都必须要依据某些数据来判断程序该如何进行。举例来说，我们在上头的 ans_yn.sh 讨论输入回应的范例中不是有练习当使用者输入 Y/N 时，必须要执行不同的讯息输出吗？简单的方式可以利用 && 与 || ，但如果我还想要执行一堆指令呢？那真的得要 if then 来帮忙啰～下面我们就来聊一聊！

12.4.1 利用 if .... then

这个 if .... then 是最常见的条件判断式了～简单的说，就是当符合某个条件判断的时候， 就予以进行某项工作就是了。这个 if ... then 的判断还有多层次的情况！我们分别介绍如下：

- 单层、简单条件判断式

如果你只有一个判断式要进行，那么我们可以简单的这样看：

```
if [ 条件判断式 ]; then
	当条件判断式成立时，可以进行的指令工作内容；
fi   &lt;==将 if 反过来写，就成为 fi 啦！结束 if 之意！
```

至于条件判断式的判断方法，与前一小节的介绍相同啊！较特别的是，如果我有多个条件要判别时， 除了 ans_yn.sh 那个案例所写的，也就是“将多个条件写入一个中括号内的情况”之外， 我还可以有多个中括号来隔开喔！而括号与括号之间，则以 && 或 || 来隔开，他们的意义是：

- && 代表 AND ；
- || 代表 or ；

所以，在使用中括号的判断式中， && 及 || 就与指令下达的状态不同了。举例来说， ans_yn.sh 里面的判断式可以这样修改：

> ["${yn}" == "Y" -o "${yn}" == "y"] 上式可替换为 ["${yn}" == "Y"] || ["${yn}" == "y"]

之所以这样改，很多人是习惯问题！很多人则是喜欢一个中括号仅有一个判别式的原因。好了， 现在我们来将 ans_yn.sh 这个脚本修改成为 if ... then 的样式来看看：

```
[dmtsai@study bin]$ cp ans_yn.sh ans_yn-2.sh  &lt;==用复制来修改的比较快！
[dmtsai@study bin]$ vim ans_yn-2.sh
#!/bin/bash
# Program:
#       This program shows the user's choice
# History:
# 2015/07/16    VBird   First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

read -p "Please input （Y/N）: " yn

if [ "${yn}" == "Y" ] &#124;&#124; [ "${yn}" == "y" ]; then
    echo "OK, continue"
    exit 0
fi
if [ "${yn}" == "N" ] &#124;&#124; [ "${yn}" == "n" ]; then
    echo "Oh, interrupt!"
    exit 0
fi
echo "I don't know what your choice is" && exit 0
```

不过，由这个例子看起来，似乎也没有什么了不起吧？原本的 ans_yn.sh 还比较简单呢～ 但是如果以逻辑概念来看，其实上面的范例中，我们使用了两个条件判断呢！明明仅有一个 ${yn} 的变量，为何需要进行两次比对呢？ 此时，多重条件判断就能够来测试测试啰！

- 多重、复杂条件判断式

在同一个数据的判断中，如果该数据需要进行多种不同的判断时，应该怎么作？举例来说，上面的 ans_yn.sh 脚本中，我们只要进行一次 ${yn} 的判断就好 （仅进行一次 if ），不想要作多次 if 的判断。 此时你就得要知道下面的语法了：

```
# 一个条件判断，分成功进行与失败进行 （else）
if [ 条件判断式 ]; then
    当条件判断式成立时，可以进行的指令工作内容；
else
    当条件判断式不成立时，可以进行的指令工作内容；
fi
```

如果考虑更复杂的情况，则可以使用这个语法：

```
# 多个条件判断 （if ... elif ... elif ... else） 分多种不同情况执行
if [ 条件判断式一 ]; then
    当条件判断式一成立时，可以进行的指令工作内容；
elif [ 条件判断式二 ]; then
    当条件判断式二成立时，可以进行的指令工作内容；
else
    当条件判断式一与二均不成立时，可以进行的指令工作内容；
fi
```

你得要注意的是， elif 也是个判断式，因此出现 elif 后面都要接 then 来处理！但是 else 已经是最后的没有成立的结果了， 所以 else 后面并没有 then 喔！好！我们来将 ans_yn-2.sh 改写成这样：

```
[dmtsai@study bin]$ cp ans_yn-2.sh ans_yn-3.sh
[dmtsai@study bin]$ vim ans_yn-3.sh
#!/bin/bash
# Program:
#       This program shows the user's choice
# History:
# 2015/07/16    VBird   First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

read -p "Please input （Y/N）: " yn

if [ "${yn}" == "Y" ] &#124;&#124; [ "${yn}" == "y" ]; then
    echo "OK, continue"
elif [ "${yn}" == "N" ] &#124;&#124; [ "${yn}" == "n" ]; then
    echo "Oh, interrupt!"
else
    echo "I don't know what your choice is"
fi
```

是否程序变得很简单，而且依序判断，可以避免掉重复判断的状况，这样真的很容易设计程序的啦！^_^！ 好了，让我们再来进行另外一个案例的设计。一般来说，如果你不希望使用者由键盘输入额外的数据时， 可以使用上一节提到的参数功能（$1）！让使用者在下达指令时就将参数带进去！ 现在我们想让使用者输入“ hello ”这个关键字时，利用参数的方法可以这样依序设计：

1. 判断 $1 是否为 hello，如果是的话，就显示 "Hello, how are you ?"；
2. 如果没有加任何参数，就提示使用者必须要使用的参数下达法；
3. 而如果加入的参数不是 hello ，就提醒使用者仅能使用 hello 为参数。

整个程序的撰写可以是这样的：

```
[dmtsai@study bin]$ vim hello-2.sh
#!/bin/bash
# Program:
#    Check $1 is equal to "hello"
# History:
# 2015/07/16    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

if [ "${1}" == "hello" ]; then
    echo "Hello, how are you ?"
elif [ "${1}" == "" ]; then
    echo "You MUST input parameters, ex&gt; {${0} someword}"
else
    echo "The only parameter is 'hello', ex&gt; {${0} hello}"
fi
```

然后你可以执行这支程序，分别在 $1 的位置输入 hello, 没有输入与随意输入， 就可以看到不同的输出啰～是否还觉得挺简单的啊！^_^。事实上， 学到这里，也真的很厉害了～好了，下面我们继续来玩一些比较大一点的计划啰～

我们在第十章已经学会了 grep 这个好用的玩意儿，那么多学一个叫做 netstat 的指令，这个指令可以查询到目前主机有打开的网络服务端口 （service ports）， 相关的功能我们会在服务器架设篇继续介绍，这里你只要知道，我可以利用“ netstat -tuln ”来取得目前主机有启动的

服务， 而且取得的信息有点像这样：

```
[dmtsai@study ~]$ netstat -tuln
Active Internet connections （only servers）
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
tcp6       0      0 ::1:25                  :::*                    LISTEN
udp        0      0 0.0.0.0:123             0.0.0.0:*
udp        0      0 0.0.0.0:5353            0.0.0.0:*
udp        0      0 0.0.0.0:44326           0.0.0.0:*
udp        0      0 127.0.0.1:323           0.0.0.0:*
udp6       0      0 :::123                  :::*
udp6       0      0 ::1:323                 :::*
#封包格式           本地IP:端口             远端IP:端口             是否监听
```

上面的重点是“Local Address （本地主机的IP与端口对应）”那个字段，他代表的是本机所启动的网络服务！ IP的部分说明的是该服务位于那个接口上，若为 127.0.0.1 则是仅针对本机开放，若是 0.0.0.0 或 ::: 则代表对整个 Internet 开放 （更多信息请参考服务器架设篇的介绍）。 每个端口 （port） 都有其特定的网络服务，几个常见的 port 与相关网络服务的关系是：

- 80: WWW
- 22: ssh
- 21: ftp
- 25: mail
- 111: RPC（远端程序调用）
- 631: CUPS（打印服务功能）

假设我的主机有兴趣要侦测的是比较常见的 port 21, 22, 25及 80 时，那我如何通过 netstat 去侦测我的主机是否有打开这四个主要的网络服务端口呢？由于每个服务的关键字都是接在冒号“ : ”后面， 所以可以借由撷取类似“ :80 ”来侦测的！那我就可以简单的这样去写这个程序喔：

```
[dmtsai@study bin]$ vim netstat.sh
#!/bin/bash
# Program:
#     Using netstat and grep to detect WWW,SSH,FTP and Mail services.
# History:
# 2015/07/16    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

# 1\. 先作一些告知的动作而已～
echo "Now, I will detect your Linux server's services!"
echo -e "The www, ftp, ssh, and mail（smtp） will be detect! \n"

# 2\. 开始进行一些测试的工作，并且也输出一些信息啰！
testfile=/dev/shm/netstat_checking.txt
netstat -tuln &gt; ${testfile}          # 先转存数据到内存当中！不用一直执行 netstat
testing=$（grep ":80 " ${testfile}）   # 侦测看 port 80 在否？
if [ "${testing}" != "" ]; then
    echo "WWW is running in your system."
fi
testing=$（grep ":22 " ${testfile}）   # 侦测看 port 22 在否？
if [ "${testing}" != "" ]; then
    echo "SSH is running in your system."
fi
testing=$（grep ":21 " ${testfile}）   # 侦测看 port 21 在否？
if [ "${testing}" != "" ]; then
    echo "FTP is running in your system."
fi
testing=$（grep ":25 " ${testfile}）   # 侦测看 port 25 在否？
if [ "${testing}" != "" ]; then
    echo "Mail is running in your system."
fi
```

实际执行这支程序你就可以看到你的主机有没有启动这些服务啦！是否很有趣呢？条件判断式还可以搞的更复杂！举例来说，在台湾当兵是国民应尽的义务，不过，在当兵的时候总是很想要退伍的！那你能不能写个脚本程序来跑，让使用者输入他的退伍日期，让你去帮他计算还有几天才退伍？

由于日期是要用相减的方式来处置，所以我们可以通过使用 date 显示日期与时间，将他转为由 1970-01-01 累积而来的秒数，通过秒数相减来取得剩余的秒数后，再换算为日数即可。整个脚本的制作流程有点像这样：

1. 先让使用者输入他们的退伍日期；
2. 再由现在日期比对退伍日期；
3. 由两个日期的比较来显示“还需要几天”才能够退伍的字样。

似乎挺难的样子？其实也不会啦，利用“ date --date="YYYYMMDD" +%s ”转成秒数后，接下来的动作就容易的多了！如果你已经写完了程序，对照下面的写法试看看：

```
[dmtsai@study bin]$ vim cal_retired.sh
#!/bin/bash
# Program:
#    You input your demobilization date, I calculate how many days before you demobilize.
# History:
# 2015/07/16    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

# 1\. 告知使用者这支程序的用途，并且告知应该如何输入日期格式？
echo "This program will try to calculate :"
echo "How many days before your demobilization date..."
read -p "Please input your demobilization date （YYYYMMDD ex&gt;20150716）: " date2

# 2\. 测试一下，这个输入的内容是否正确？利用正则表达式啰～
date_d=$（echo ${date2} &#124;grep '[0-9]\{8\}'）   # 看看是否有八个数字
if [ "${date_d}" == "" ]; then
    echo "You input the wrong date format...."
    exit 1
fi

# 3\. 开始计算日期啰～
declare -i date_dem=$（date --date="${date2}" +%s）      # 退伍日期秒数
declare -i date_now=$（date +%s）                        # 现在日期秒数
declare -i date_total_s=$（（${date_dem}-${date_now}））   # 剩余秒数统计
declare -i date_d=$（（${date_total_s}/60/60/24））         # 转为日数
if [ "${date_total_s}" -lt "0" ]; then                  # 判断是否已退伍
    echo "You had been demobilization before: " $（（-1*${date_d}）） " ago"
else
    declare -i date_h=$（（$（（${date_total_s}-${date_d}*60*60*24））/60/60））
    echo "You will demobilize after ${date_d} days and ${date_h} hours."
fi
```

瞧一瞧，这支程序可以帮你计算退伍日期呢～如果是已经退伍的朋友， 还可以知道已经退伍多久了～哈哈！很可爱吧～脚本中的 dated 变量宣告那个 */60/60/24* 是来自于一天的总秒数（*24*小时*60*分*60*秒） 。瞧～全部的动作都没有超出我们所学的范围吧～ ^^ 还能够避免使用者输入错误的数字，所以多了一个正则表达式的判断式呢～ 这个例子比较难，有兴趣想要一探究竟的朋友，可以作一下课后练习题 关于计算生日的那一题喔！～加油！

12.4.2 利用 case ..... esac 判断

上个小节提到的“ if .... then .... fi ”对于变量的判断是以“比对”的方式来分辨的， 如果符合状态就进行某些行为，并且通过较多层次 （就是 elif ...） 的方式来进行多个变量的程序码撰写，譬如 hello-2.sh 那个小程序，就是用这样的方式来撰写的啰。 好，那么万一我有多个既定的变量内容，例如 hello-2.sh 当中，我所需要的变量就是 "hello" 及空字串两个， 那么我只要针对这两个变量来设置状况就好了，对吧？那么可以使用什么方式来设计呢？呵呵～就用 case ... in .... esac 吧～，他的语法如下：

```
case  $变量名称 in   &lt;==关键字为 case ，还有变量前有钱字号
  "第一个变量内容")   &lt;==每个变量内容建议用双引号括起来，关键字则为小括号 ）
    程序段
    ;;            &lt;==每个类别结尾使用两个连续的分号来处理！
  "第二个变量内容")
    程序段
    ;;
  *)                  &lt;==最后一个变量内容都会用 * 来代表所有其他值
    不包含第一个变量内容与第二个变量内容的其他程序执行段
    exit 1
    ;;
esac                  &lt;==最终的 case 结尾！“反过来写”思考一下！
```

要注意的是，这个语法以 case （实际案例之意） 为开头，结尾自然就是将 case 的英文反过来写！就成为 esac 啰！ 不会很难背啦！另外，每一个变量内容的程序段最后都需要两个分号 （;;） 来代表该程序段落的结束，这挺重要的喔！ 至于为何需要有 * 这个变量内容在最后呢？这是因为，如果使用者不是输入变量内容一或二时， 我们可以告知使用者相关的信息啊！废话少说，我们拿 hello-2.sh 的案例来修改一下，他应该会变成这样喔：

```
[dmtsai@study bin]$ vim hello-3.sh
#!/bin/bash
# Program:
#     Show "Hello" from $1.... by using case .... esac
# History:
# 2015/07/16    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

case ${1} in
  "hello")
    echo "Hello, how are you ?"
    ;;
  "")
    echo "You MUST input parameters, ex&gt; {${0} someword}"
    ;;
  *)   # 其实就相当于万用字符，0~无穷多个任意字符之意！
    echo "Usage ${0} {hello}"
    ;;
esac
```

在上面这个 hello-3.sh 的案例当中，如果你输入“ sh hello-3.sh test ”来执行， 那么屏幕上就会出现“Usage hello-3.sh {hello}”的字样，告知执行者仅能够使用 hello 喔～ 这样的方式对于需要某些固定字串来执行的变量内容就显的更加的方便呢！ 这种方式你真的要熟悉喔！这是因为早期系统的很多服务的启动 scripts 都是使用这种写法的 （CentOS 6.x 以前）。 虽然 CentOS 7 已经使用 systemd，不过仍有数个服务是放在 /etc/init.d/ 目录下喔！例如有个名为 netconsole 的服务在该目录下， 那么你想要重新启动该服务，是可以这样做的 （请注意，要成功执行，还是得要具有 root 身份才行！一般帐号能执行，但不会成功！）：

> /etc/init.d/netconsole restart

重点是那个 restart 啦！如果你使用“ less /etc/init.d/netconsole ”去查阅一下，就会看到他使用的是 case 语法， 并且会规定某些既定的变量内容，你可以直接下达 /etc/init.d/netconsole ，该 script 就会告知你有哪些后续接的变量可以使用啰～方便吧！ ^_^

一般来说，使用“ case $变量 in ”这个语法中，当中的那个“ $变量 ”大致有两种取得的方式：

- 直接下达式：例如上面提到的，利用“ script.sh variable ” 的方式来直接给予 $1 这个变量的内容，这也是在 /etc/init.d 目录下大多数程序的设计方式。
- 互动式：通过 read 这个指令来让使用者输入变量的内容。

这么说或许你的感受性还不高，好，我们直接写个程序来玩玩：让使用者能够输入 one, two, three ， 并且将使用者的变量显示到屏幕上，如果不是 one, two, three 时，就告知使用者仅有这三种选择。

```
[dmtsai@study bin]$ vim show123.sh
#!/bin/bash
# Program:
#    This script only accepts the flowing parameter: one, two or three.
# History:
# 2015/07/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

echo "This program will print your selection !"
# read -p "Input your choice: " choice   # 暂时取消，可以替换！
# case ${choice} in                      # 暂时取消，可以替换！
case ${1} in                             # 现在使用，可以用上面两行替换！
  "one"）
    echo "Your choice is ONE"
    ;;
  "two"）
    echo "Your choice is TWO"
    ;;
  "three"）
    echo "Your choice is THREE"
    ;;
  *）
    echo "Usage ${0} {one&#124;two&#124;three}"
    ;;
esac
```

此时，你可以使用“ sh show123.sh two ”的方式来下达指令，就可以收到相对应的回应了。上面使用的是直接下达的方式，而如果使用的是互动式时，那么将上面第 10, 11 行的 "#" 拿掉， 并将 12 行加上注解 （#），就可以让使用者输入参数啰～这样是否很有趣啊？

12.4.3 利用 function 功能

什么是“函数 （function）”功能啊？简单的说，其实， 函数可以在 shell script 当中做出一个类似自订执行指令的东西，最大的功能是， 可以简化我们很多的程序码～举例来说，上面的 show123.sh 当中，每个输入结果 one, two, three 其实输出的内容都一样啊～那么我就可以使用 function 来简化了！ function 的语法是这样的：

```
function fname（） {
    程序段
}
```

那个 fname 就是我们的自订的执行指令名称～而程序段就是我们要他执行的内容了。 要注意的是，因为 shell script 的执行方式是由上而下，由左而右， 因此在 shell script 当中的 function 的设置一定要在程序的最前面， 这样才能够在执行时被找到可用的程序段喔 （这一点与传统程序语言差异相当大！初次接触的朋友要小心！）！ 好～我们将 show123.sh 改写一下，自订一个名为 printit 的函数来使用喔：

```
[dmtsai@study bin]$ vim show123-2.sh
#!/bin/bash
# Program:
#    Use function to repeat information.
# History:
# 2015/07/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

function printit（）{
    echo -n "Your choice is "     # 加上 -n 可以不断行继续在同一行显示
}

echo "This program will print your selection !"
case ${1} in
  "one"）
    **printit**; echo ${1} &#124; tr 'a-z' 'A-Z'  # 将参数做大小写转换！
    ;;
  "two"）
    **printit**; echo ${1} &#124; tr 'a-z' 'A-Z'
    ;;
  "three"）
    **printit**; echo ${1} &#124; tr 'a-z' 'A-Z'
    ;;
  *）
    echo "Usage ${0} {one&#124;two&#124;three}"
    ;;
esac
```

以上面的例子来说，鸟哥做了一个函数名称为 printit ，所以，当我在后续的程序段里面， 只要执行 printit 的话，就表示我的 shell script 要去执行“ function printit .... ” 里面的那几个程序段落啰！当然啰，上面这个例子举得太简单了，所以你不会觉得 function 有什么好厉害的， 不过，如果某些程序码一再地在 script 当中重复时，这个 function 可就重要的多啰～ 不但可以简化程序码，而且可以做成类似“模块”的玩意儿，真的很棒啦！



Tips 建议读者可以使用类似 vim 的编辑器到 /etc/init.d/ 目录下去查阅一下你所看到的文件， 并且自行追踪一下每个文件的执行情况，相信会更有心得！

另外， function 也是拥有内置变量的～他的内置变量与 shell script 很类似， 函数名称代表示 $0 ，而后续接的变量也是以 $1, $2... 来取代的～ 这里很容易搞错喔～因为“ function fname（） { 程序段 } ”内的 $0, $1... 等等与 shell script 的 $0 是不同的。以上面 show123-

2.sh 来说，假如我下达：" sh show123-2.sh one " 这表示在 shell script 内的 $1 为 "one" 这个字串。但是在 printit（） 内的 $1 则与这个 one 无关。 我们将上面的例子再次的改写一下，让你更清楚！

```
[dmtsai@study bin]$ vim show123-3.sh
#!/bin/bash
# Program:
#    Use function to repeat information.
# History:
# 2015/07/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

function printit（）{
    echo "Your choice is ${1}"   # 这个 $1 必须要参考下面指令的下达
}

echo "This program will print your selection !"
case ${1} in
  "one"）
    **printit 1**  # 请注意， printit 指令后面还有接参数！
    ;;
  "two"）
    **printit 2**
    ;;
  "three"）
    **printit 3**
    ;;
  *）
    echo "Usage ${0} {one&#124;two&#124;three}"
    ;;
esac
```

在上面的例子当中，如果你输入" sh show123-3.sh one "就会出现" Your choice is 1 "的字样～ 为什么是 1 呢？因为在程序段落当中，我们是写了" printit 1 "那个 1 就会成为 function 当中的 $1 喔～ 这样是否理解呢？ function 本身其实比较困难一点，如果你还想要进行其他的撰写的话。 不过，我们仅是想要更加了解 shell script 而已，所以，这里看看即可～了解原理就好啰～ ^_^

12.5 循环（loop）

除了 if...then...fi 这种条件判断式之外，循环可能是程序当中最重要的一环了～ 循环可以不断的执行某个程序段落，直到使用者设置的条件达成为止。 所以，重点是那个“条件的达成”是什么。除了这种依据判断式达成与否的不定循环之外， 还有另外一种已经固定要跑多少次的循环形态，可称为固定循环的形态呢！下面我们就来谈一谈：

12.5.1 while do done, until do done （不定循环）

一般来说，不定循环最常见的就是下面这两种状态了：

```
while [ condition ]  &lt;==中括号内的状态就是判断式
do            &lt;==do 是循环的开始！
    程序段落
done          &lt;==done 是循环的结束
```

while 的中文是“当....时”，所以，这种方式说的是“当 condition 条件成立时，就进行循环，直到 condition 的条件不成立才停止”的意思。还有另外一种不定循环的方式：

```
until [ condition ]
do
    程序段落
done
```

这种方式恰恰与 while 相反，它说的是“当 condition 条件成立时，就终止循环， 否则就持续进行循环的程序段。”是否刚好相反啊～我们以 while 来做个简单的练习好了。 假设我要让使用者输入 yes 或者是 YES 才结束程序的执行，否则就一直进行告知使用者输入字串。

```
[dmtsai@study bin]$ vim yes_to_stop.sh
#!/bin/bash
# Program:
#    Repeat question until user input correct answer.
# History:
# 2015/07/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

while [ "${yn}" != "yes" -a "${yn}" != "YES" ]
do
    read -p "Please input yes/YES to stop this program: " yn
done
echo "OK! you input the correct answer."
```

上面这个例题的说明是“当 ${yn} 这个变量不是 "yes" 且 ${yn} 也不是 "YES" 时，才进行循环内的程序。” 而如果 ${yn} 是 "yes" 或 "YES" 时，就会离开循环啰～那如果使用 until 呢？呵呵有趣啰～ 他的条件会变成这样：

```
[dmtsai@study bin]$ vim yes_to_stop-2.sh
#!/bin/bash
# Program:
#    Repeat question until user input correct answer.
# History:
# 2015/07/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

until [ "${yn}" == "yes" -o "${yn}" == "YES" ]
do
    read -p "Please input yes/YES to stop this program: " yn
done
echo "OK! you input the correct answer."
```

仔细比对一下这两个东西有啥不同喔！ ^_^再来，如果我想要计算 1+2+3+....+100 这个数据呢？ 利用循环啊～他是这样的：

```
[dmtsai@study bin]$ vim cal_1_100.sh
#!/bin/bash
# Program:
#    Use loop to calculate "1+2+3+...+100" result.
# History:
# 2015/07/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

s=0  # 这是加总的数值变量
i=0  # 这是累计的数值，亦即是 1, 2, 3....
while [ "${i}" != "100" ]
do
    i=$（（$i+1））   # 每次 i 都会增加 1
    s=$（（$s+$i））  # 每次都会加总一次！
done
echo "The result of '1+2+3+...+100' is ==&gt; $s"
```

嘿嘿！当你执行了“ sh cal_1_100.sh ”之后，就可以得到 5050 这个数据才对啊！这样瞭呼～那么让你自行做一下，如果想要让使用者自行输入一个数字，让程序由 1+2+... 直到你输入的数字为止， 该如何撰写呢？应该很简单吧？答案可以参考一下习题练习里面的一题喔！

12.5.2 for...do...done （固定循环）

相对于 while, until 的循环方式是必须要“符合某个条件”的状态， for 这种语法，则是“ 已经知道要进行几次循环”的状态！他的语法是：

```
for var in con1 con2 con3 ...
do
    程序段
done
```

以上面的例子来说，这个 $var 的变量内容在循环工作时：

1. 第一次循环时， $var 的内容为 con1 ；
2. 第二次循环时， $var 的内容为 con2 ；
3. 第三次循环时， $var 的内容为 con3 ；

4. ....

我们可以做个简单的练习。假设我有三种动物，分别是 dog, cat, elephant 三种， 我想每一行都输出这样："There are dogs..."之类的字样，则可以：

```
[dmtsai@study bin]$ vim show_animal.sh
#!/bin/bash
# Program:
#     Using for .... loop to print 3 animals
# History:
# 2015/07/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

for animal in dog cat elephant
do
    echo "There are ${animal}s.... "
done
```

等你执行之后就能够发现这个程序运行的情况啦！让我们想像另外一种状况，由于系统上面的各种帐号都是写在 /etc/passwd 内的第一个字段，你能不能通过管线命令的 cut 捉出单纯的帐号名称后，以 id 分别检查使用者的识别码与特殊参数呢？由于不同的 Linux 系统上面的帐号都不一样！此时实际去捉 /etc/passwd 并使用循环处理，就是一个可行的方案了！程序可以如下：

```
[dmtsai@study bin]$ vim userid.sh
#!/bin/bash
# Program
#       Use id, finger command to check system account's information.
# History
# 2015/07/17    VBird   first release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
users=$（cut -d ':' -f1 /etc/passwd）    # 撷取帐号名称
for username in ${users}                # 开始循环进行！
do
        id ${username}
done
```

执行上面的脚本后，你的系统帐号就会被捉出来检查啦！这个动作还可以用在每个帐号的删除、重整上面呢！ 换个角度来看，如果我现在需要一连串的数字来进行循环呢？举例来说，我想要利用 ping 这个可以判断网络状态的指令， 来进行网络状态的实际侦测时，我想要侦测的网域是本机所在的 192.168.1.1~192.168.1.100，由于有 100 台主机， 总不会要我在 for 后面输入 1 到 100 吧？此时你可以这样做喔！

```
[dmtsai@study bin]$ vim pingip.sh
#!/bin/bash
# Program
#       Use ping command to check the network's PC state.
# History
# 2015/07/17    VBird   first release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
network="192.168.1"              # 先定义一个网域的前面部分！
for sitenu in $（seq 1 100）       # seq 为 sequence（连续） 的缩写之意
do
    # 下面的程序在取得 ping 的回传值是正确的还是失败的！
        ping -c 1 -w 1 ${network}.${sitenu} &&gt; /dev/null && result=0 &#124;&#124; resu
    # 开始显示结果是正确的启动 （UP） 还是错误的没有连通 （DOWN）
        if [ "${result}" == 0 ]; then
                echo "Server ${network}.${sitenu} is UP."
        else
                echo "Server ${network}.${sitenu} is DOWN."
        fi
done
```

上面这一串指令执行之后就可以显示出 192.168.1.1~192.168.1.100 共 100 部主机目前是否能与你的机器连通！ 如果你的网域与鸟哥所在的位置不同，则直接修改上头那个 network 的变量内容即可！其实这个范例的重点在 $（seq ..） 那个位置！那个 seq 是连续（sequence） 的缩写之意！代表后面接的两个数值是一直连续的！ 如此一来，就能够轻松的将连续数字带入程序中啰！



Tips 除了使用 $（seq 1 100） 之外，你也可以直接使用 bash 的内置机制来处理喔！可以使用 {1..100} 来取代 $（seq 1 100） ！ 那个大括号内的前面/后面用两个字符，中间以两个小数点来代表连续出现的意思！例如要持续输出 a, b, c...g 的话， 就可以使用“ echo {a..g} ”这样的表示方式！

最后，让我们来玩判断式加上循环的功能！我想要让使用者输入某个目录文件名， 然后我找出某目录内的文件名的权限，该如何是好？呵呵！可以这样做啦～

```
[dmtsai@study bin]$ vim dir_perm.sh
#!/bin/bash
# Program:
#    User input dir name, I find the permission of files.
# History:
# 2015/07/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

# 1\. 先看看这个目录是否存在啊？
read -p "Please input a directory: " dir
if [ "${dir}" == "" -o ! -d "${dir}" ]; then
    echo "The ${dir} is NOT exist in your system."
    exit 1
fi

# 2\. 开始测试文件啰～
filelist=$（ls ${dir}）        # 列出所有在该目录下的文件名称
for filename in ${filelist}
do
    perm=""
    test -r "${dir}/${filename}" && perm="${perm} readable"
    test -w "${dir}/${filename}" && perm="${perm} writable"
    test -x "${dir}/${filename}" && perm="${perm} executable"
    echo "The file ${dir}/${filename}'s permission is ${perm} "
done
```

呵呵！很有趣的例子吧～利用这种方式，你可以很轻易的来处理一些文件的特性呢。接下来，让我们来玩玩另一种 for 循环的功能吧！主要用在数值方面的处理喔！

12.5.3 for...do...done 的数值处理

除了上述的方法之外，for 循环还有另外一种写法！语法如下：

```
for （（ 初始值; 限制值; 执行步阶 ））
do
    程序段
done
```

这种语法适合于数值方式的运算当中，在 for 后面的括号内的三串内容意义为：

- 初始值：某个变量在循环当中的起始值，直接以类似 i=1 设置好；
- 限制值：当变量的值在这个限制值的范围内，就继续进行循环。例如 i<=100；
- 执行步阶：每作一次循环时，变量的变化量。例如 i=i+1。

值得注意的是，在“执行步阶”的设置上，如果每次增加 1 ，则可以使用类似“i++”的方式，亦即是 i 每次循环都会增加一的意思。好，我们以这种方式来进行 1 累加到使用者输入的循环吧！

```
[dmtsai@study bin]$ vim cal_1_100-2.sh
#!/bin/bash
# Program:
#     Try do calculate 1+2+....+${your_input}
# History:
# 2015/07/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

read -p "Please input a number, I will count for 1+2+...+your_input: " nu

s=0
for （（ i=1; i&lt;=${nu}; i=i+1 ））
do
    s=$（（${s}+${i}））
done
echo "The result of '1+2+3+...+${nu}' is ==&gt; ${s}"
```

一样也是很简单吧！利用这个 for 则可以直接限制循环要进行几次呢！

12.5.4 搭配乱数与阵列的实验

现在你大概已经能够掌握 shell script 了！好了！让我们来做个小实验！假设你们公司的团队中，经常为了今天中午要吃啥搞到头很昏！ 每次都用猜拳的～好烦喔～有没有办法写支脚本，用脚本搭配乱数来告诉我们，今天中午吃啥好？呵呵！执行这只脚本后， 直接跟你说要吃啥～那比猜拳好多了吧？哈哈！

要达成这个任务，首先你得要将全部的店家输入到一组阵列当中，再通过乱数的处理，去取得可能的数值，再将搭配到该数值的店家秀出来即可！ 其实也很简单！让我们来实验看看：

```
[dmtsai@study bin]$ vim what_to_eat.sh
#!/bin/bash
# Program:
#     Try do tell you what you may eat.
# History:
# 2015/07/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

eat[1]="卖当当漢堡包"       # 写下你所收集到的店家！
eat[2]="肯爷爷炸鸡"
eat[3]="彩虹日式便当"
eat[4]="越油越好吃大雅"
eat[5]="想不出吃啥学餐"
eat[6]="太师父便当"
eat[7]="池上便当"
eat[8]="怀念火车便当"
eat[9]="一起吃方便面"
eatnum=9                  # 需要输入有几个可用的餐厅数！

check=$（（ ${RANDOM} * ${eatnum} / 32767 + 1 ））
echo "your may eat ${eat[${check}]}"
```

立刻执行看看，你就知道该吃啥了！非常有趣吧！不过，这个例子中只选择一个样本，不够看！如果想要每次都秀出 3 个店家呢？ 而且这个店家不能重复喔！重复当然就没啥意义了！所以，你可以这样作！

```
[dmtsai@study bin]$ vim what_to_eat-2.sh
#!/bin/bash
# Program:
#     Try do tell you what you may eat.
# History:
# 2015/07/17    VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

eat[1]="卖当当漢堡包"
eat[2]="肯爷爷炸鸡"
eat[3]="彩虹日式便当"
eat[4]="越油越好吃大雅"
eat[5]="想不出吃啥学餐"
eat[6]="太师父便当"
eat[7]="池上便当"
eat[8]="怀念火车便当"
eat[9]="一起吃方便面"
eatnum=9

eated=0
while [ "${eated}" -lt 3 ]; do
        check=$（（ ${RANDOM} * ${eatnum} / 32767 + 1 ））
        mycheck=0
        if [ "${eated}" -ge 1 ]; then
                for i in $（seq 1 ${eated} ）
                do
                        if [ ${eatedcon[$i]} == $check ]; then
                                mycheck=1
                        fi
                done
        fi
        if [ ${mycheck} == 0 ]; then
                echo "your may eat ${eat[${check}]}"
                eated=$（（ ${eated} + 1 ））
                eatedcon[${eated}]=${check}
        fi
done
```

通过乱数、阵列、循环与条件判断，你可以做出很多很特别的东西！还不用写传统程序语言～试看看～挺有趣的呦！

12.6 shell script 的追踪与 debug

scripts 在执行之前，最怕的就是出现语法错误的问题了！那么我们如何 debug 呢？有没有办法不需要通过直接执行该 scripts 就可以来判断是否有问题呢？呵呵！当然是有的！我们就直接以 bash 的相关参数来进行判断吧！

```
[dmtsai@study ~]$ sh [-nvx] scripts.sh
选项与参数：
-n  ：不要执行 script，仅查询语法的问题；
-v  ：再执行 sccript 前，先将 scripts 的内容输出到屏幕上；
-x  ：将使用到的 script 内容显示到屏幕上，这是很有用的参数！

范例一：测试 dir_perm.sh 有无语法的问题？
[dmtsai@study ~]$ sh -n dir_perm.sh
# 若语法没有问题，则不会显示任何信息！

范例二：将 show_animal.sh 的执行过程全部列出来～
[dmtsai@study ~]$ sh -x show_animal.sh
+ PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/root/bin
+ export PATH
+ for animal in dog cat elephant
+ echo 'There are dogs.... '
There are dogs....
+ for animal in dog cat elephant
+ echo 'There are cats.... '
There are cats....
+ for animal in dog cat elephant
+ echo 'There are elephants.... '
There are elephants....
```

请注意，上面范例二中执行的结果并不会有颜色的显示！鸟哥为了方便说明所以在 + 号之后的数据都加上颜色了！ 在输出的讯息中，在加号后面的数据其实都是指令串，由于 sh -x 的方式来将指令执行过程也显示出来， 如此使用者可以判断程序码执行到哪一段时会出现相关的信息！这个功能非常的棒！通过显示完整的指令串， 你就能够依据输出的错误信息来订正你的脚本了！

熟悉 sh 的用法，将可以使你在管理 Linux 的过程中得心应手！至于在 Shell scripts 的学习方法上面，需要“多看、多模仿、并加以修改成自己的样式！”是最快的学习手段了！网络上有相当多的朋友在开发一些相当有用的 scripts ，若是你可以将对方的 scripts 拿来，并且改成适合自己主机的样子！那么学习的效果会是最快的呢！

另外，我们 Linux 系统本来就有很多的服务启动脚本，如果你想要知道每个 script 所代表的功能是什么？ 可以直接以 vim 进入该 script 去查阅一下，通常立刻就知道该 script 的目的了。 举例来说，我们之前一直提到的 /etc/init.d/netconsole ，这个 script 是干嘛用的？ 利用 vim 去查阅最前面的几行字，他出现如下信息：

```
# netconsole    This loads the netconsole module with the configured parameters.
# chkconfig: - 50 50
# description: Initializes network console logging
# config: /etc/sysconfig/netconsole
```

意思是说，这个脚本在设置网络终端机来应付登陆的意思，且配置文件在 /etc/sysconfig/netconsole 设置内！ 所以，你写的脚本如果也能够很清楚的交待，那就太棒了！

另外，本章所有的范例都可以在 http://linux.vbird.org/linux_basic/0340bashshell-scripts/scripts-20150717.tar.bz2 里头找到喔！加油～

12.7 重点回顾

- shell script 是利用 shell 的功能所写的一个“程序 （program）”，这个程序是使用纯文本文件，将一些 shell 的语法与指令（含外部指令）写在里面， 搭配正则表达式、管线命令与数据流重导向等功能，以达到我们所想要的处理目的
- shell script 用在系统管理上面是很好的一项工具，但是用在处理大量数值运算上， 就不够好了，因为 Shell scripts 的速度较慢，且使用的 CPU 资源较多，造成主机资源的分配不良。
- 在 Shell script 的文件中，指令的执行是从上而下、从左而右的分析与执行；
- shell script 的执行，至少需要有 r 的权限，若需要直接指令下达，则需要拥有 r 与 x 的权限；
- 良好的程序撰写习惯中，第一行要宣告 shell （#!/bin/bash） ，第二行以后则宣告程序用途、版本、作者等
- 对谈式脚本可用 read 指令达成；
- 要创建每次执行脚本都有不同结果的数据，可使用 date 指令利用日期达成；
- script 的执行若以 source 来执行时，代表在父程序的 bash 内执行之意！
- 若需要进行判断式，可使用 test 或中括号 （ [] ） 来处理；
- 在 script 内，$0, $1, $2..., $@ 是有特殊意义的！
- 条件判断式可使用 if...then 来判断，若是固定变量内容的情况下，可使用 case $var in ... esac 来处理
- 循环主要分为不定循环 （while, until） 以及固定循环 （for） ，配合 do, done 来达成所需任务！
- 我们可使用 sh -x script.sh 来进行程序的 debug

12.8 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ） 下面皆为实作题，请自行撰写出程序喔！

- 请创建一支 script ，当你执行该 script 的时候，该 script 可以显示： 1. 你目前的身份 （用 whoami ） 2. 你目前所在的目录 （用 pwd）

```
#!/bin/bash
echo -e "Your name is ==&gt; $（whoami）"
echo -e "The current directory is ==&gt; $（pwd）"
```

- 请自行创建一支程序，该程序可以用来计算“你还有几天可以过生日”啊？

```
#!/bin/bash
read -p "Pleas input your birthday （MMDD, ex&gt; 0709）: " bir
now=`date +%m%d`
if [ "$bir" == "$now" ]; then
echo "Happy Birthday to you!!!"
elif [ "$bir" -gt "$now" ]; then
year=`date +%Y`
total_d=$（（$（（`date --date="$year$bir" +%s`-`date +%s`））/60/60/24））
echo "Your birthday will be $total_d later"
else
year=$（（`date +%Y`+1））
total_d=$（（$（（`date --date="$year$bir" +%s`-`date +%s`））/60/60/24））
echo "Your birthday will be $total_d later"
fi
```

- 让使用者输入一个数字，程序可以由 1+2+3... 一直累加到使用者输入的数字为止。

```
#!/bin/bash
read -p "Please input an integer number: " number
i=0
s=0
while [ "$i" != "$number" ]
do
i=$（（$i+1））
s=$（（$s+$i））
done
echo "the result of '1+2+3+...$number' is ==&gt; $s"
```

- 撰写一支程序，他的作用是: 1.） 先查看一下 /root/test/logical 这个名称是否存在； 2.） 若不存在，则创建一个文件，使用 touch 来创建，创建完成后离开； 3.） 如果存在的话，判断该名称是否为文件，若为文件则将之删除后创建一个目录，文件名为 logical ，之后离开； 4.） 如果存在的话，而且该名称为目录，则移除此目录！

```
#!/bin/bash
if [ ! -e logical ]; then
touch logical
echo "Just make a file logical"
exit 1
elif [ -e logical ] && [ -f logical ]; then
rm logical
mkdir logical
echo "remove file ==&gt; logical"
echo "and make directory logical"
exit 1
elif [ -e logical ] && [ -d logical ]; then
rm -rf logical
echo "remove directory ==&gt; logical"
exit 1
else
echo "Does here have anything?"
fi
```

- 我们知道 /etc/passwd 里面以 : 来分隔，第一栏为帐号名称。请写一只程序，可以将 /etc/passwd 的第一栏取出，而且每一栏都以一行字串“The 1 account is "root" ”来显示，那个 1 表示行数。

```
#!/bin/bash
accounts=`cat /etc/passwd | cut -d':' -f1`
for account in $accounts
do
declare -i i=$i+1
echo "The $i account is \"$account\" "
done
```

2002/06/27：第一次完成 2003/02/10：重新编排与加入 FAQ 2005/08/29：将旧的文章移动到这里 了。 2005/08/29：呼呼～加入了一些比较有趣的练习题～比第一版要难的多～大家多多玩一玩喔～ 2009/02/10：将旧的基于 FC4 版本的文章移动到此处 2009/02/17：加入 shift 的介绍 2009/02/18：加入了一些额外的练习，包括 for 的 ping 处理！ 2015/07/17：全部的脚本通通小修改过，并且加上阵列、乱数与午餐大乱斗的脚本程序

第十三章、Linux 帐号管理与 ACL 权限设置

最近更新日期：20//

要登陆 Linux 系统一定要有帐号与密码才行，否则怎么登陆，您说是吧？不过， 不同的使用者应该要拥有不同的权限才行吧？我们还可以通过 user/group 的特殊权限设置， 来规范出不同的群组开发专案呢～在 Linux 的环境下，我们可以通过很多方式来限制使用者能够使用的系统资源， 包括 第十章、bash 提到的 ulimit 限制、还有特殊权限限制，如 umask 等等。 通过这些举动，我们可以规范出不同使用者的使用资源。另外，还记得系统管理员的帐号吗？对！ 就是 root 。请问一下，除了 root 之外，是否可以有其他的系统管理员帐号？ 为什么大家都要尽量避免使用数字体态的帐号？如何修改使用者相关的信息呢？这些我们都得要了解了解的！

13.1 Linux 的帐号与群组

管理员的工作中，相当重要的一环就是“管理帐号”啦！因为整个系统都是你在管理的， 并且所有一般用户的帐号申请，都必须要通过你的协助才行！所以你就必须要了解一下如何管理好一个服务器主机的帐号啦！ 在管理 Linux 主机的帐号时，我们必须先来了解一下 Linux 到底是如何辨别每一个使用者的！

13.1.1 使用者识别码： UID 与 GID

虽然我们登陆 Linux 主机的时候，输入的是我们的帐号，但是其实 Linux 主机并不会直接认识你的“帐号名称”的，他仅认识 ID 啊 （ID 就是一组号码啦）。 由于计算机仅认识 0 与 1，所以主机对于数字比较有概念的；至于帐号只是为了让人们容易记忆而已。 而你的 ID 与帐号的对应就在 /etc/passwd 当中哩。



Tips 如果你曾经在网络上下载过 tarball 类型的文件， 那么应该不难发现，在解压缩之后的文件中，文件拥有者的字段竟然显示“不明的数字”？奇怪吧？这没什么好奇怪的，因为 Linux 说实在话，他真的只认识代表你身份的号码而已！

那么到底有几种 ID 呢？还记得我们在第五章内有提到过， 每一个文件都具有“拥有人与拥有群组”的属性吗？没错啦～每个登陆的使用者至少都会取得两个 ID ，一个是使用者 ID （User ID ，简称 UID）、一个是群组 ID （Group ID ，简称 GID）。

那么文件如何判别他的拥有者与群组呢？其实就是利用 UID 与 GID 啦！每一个文件都会有所谓的拥有者 ID 与拥有群组 ID ，当我们有要显示文件属性的需求时，系统会依据 /etc/passwd 与 /etc/group 的内容， 找到 UID / GID 对应的帐号与群组名称再显示出来！我们可以作个小实验，你可以用 root 的身份 vim /etc/passwd ，然后将你的一般身份的使用者的 ID 随便改一个号码，然后再到你的一般身份的目录下看看原先该帐号拥有的文件，你会发现该文件的拥有人变成了 “数字了”呵呵！这样可以理解了吗？来看看下面的例子：

```
# 1\. 先察看一下，系统里面有没有一个名为 dmtsai 的用户？
[root@study ~]# id dmtsai
uid=1000（dmtsai） gid=1000（dmtsai） groups=1000（dmtsai）,10（wheel）  &lt;==确定有这个帐号喔

[root@study ~]# ll -d /home/dmtsai
drwx------. 17 dmtsai dmtsai 4096 Jul 17 19:51 /home/dmtsai
# 瞧一瞧，使用者的字段正是 dmtsai 本身喔！

# 2\. 修改一下，将刚刚我们的 dmtsai 的 1000 UID 改为 2000 看看：
[root@study ~]# vim /etc/passwd
....（前面省略）....
dmtsai:x:2000:1000:dmtsai:/home/dmtsai:/bin/bash &lt;==修改一下特殊字体部分，由 1000 改过来
[root@study ~]# ll -d /home/dmtsai
drwx------. 17 1000 dmtsai 4096 Jul 17 19:51 /home/dmtsai
# 很害怕吧！怎么变成 1000 了？因为文件只会记录 UID 的数字而已！
# 因为我们乱改，所以导致 1000 找不到对应的帐号，因此显示数字！

# 3\. 记得将刚刚的 2000 改回来！
[root@study ~]# vim /etc/passwd
....（前面省略）....
dmtsai:x:1000:1000:dmtsai:/home/dmtsai:/bin/bash  &lt;==“务必一定要”改回来！
```

你一定要了解的是，上面的例子仅是在说明 UID 与帐号的对应性，在一部正常运行的 Linux 主机环境下，上面的动作不可随便进行， 这是因为系统上已经有很多的数据被创建存在了，随意修改系统上某些帐号的 UID 很可能会导致某些程序无法进行，这将导致系统无法顺利运行的结果， 因为权限的问题啊！所以，了解了之后，请赶快回到 /etc/passwd 里面，将数字改回来喔！



Tips 举例来说，如果上面的测试最后一个步骤没有将 2000 改回原本的 UID，那么当 dmtsai 下次登陆时将没有办法进入自己的主文件夹！ 因为他的 UID 已经改为 2000 ，但是他的主文件夹 （/home/dmtsai） 却记录的是 1000 ，由于权限是 700 ， 因此他将无法进入原本的主文件夹！是否非常严重啊？

13.1.2 使用者帐号

Linux 系统上面的使用者如果需要登陆主机以取得 shell 的环境来工作时，他需要如何进行呢？ 首先，他必须要在计算机前面利用 tty1~tty6 的终端机提供的 login 接口，并输入帐号与密码后才能够登陆。 如果是通过网络的话，那至少使用者就得要学习 ssh 这个功能了 （服务器篇再来谈）。 那么你输入帐号密码后，系统帮你处理了什么呢？

1. 先找寻 /etc/passwd 里面是否有你输入的帐号？如果没有则跳出，如果有的话则将该帐号对应的 UID 与 GID （在 /etc/group 中） 读出来，另外，该帐号的主文件夹与 shell 设置也一并读出；

2. 再来则是核对密码表啦！这时 Linux 会进入 /etc/shadow 里面找出对应的帐号与 UID，然后核对一下你刚刚输入的密码与里头的密码是否相符？

3. 如果一切都 OK 的话，就进入 Shell 控管的阶段啰！

大致上的情况就像这样，所以当你要登陆你的 Linux 主机的时候，那个 /etc/passwd 与 /etc/shadow 就必须要让系统读取啦 （这也是很多攻击者会将特殊帐号写到 /etc/passwd 里头去的缘故），所以呢，如果你要备份 Linux 的系统的帐号的话，那么这两个文件就一定需要备份才行呦！

由上面的流程我们也知道，跟使用者帐号有关的有两个非常重要的文件，一个是管理使用者 UID/GID 重要参数的 /etc/passwd ，一个则是专门管理密码相关数据的 /etc/shadow 啰！那这两个文件的内容就非常值得进行研究啦！ 下面我们会简单的介绍这两个文件，详细的说明可以参考 man 5 passwd 及 man 5 shadow [1]。

- /etc/passwd 文件结构

这个文件的构造是这样的：每一行都代表一个帐号，有几行就代表有几个帐号在你的系统中！ 不过需要特别留意的是，里头很多帐号本来就是系统正常运行所必须要的，我们可以简称他为系统帐号， 例如 bin, daemon, adm, nobody 等等，这些帐号请不要随意的杀掉他呢！ 这个文件的内容有点像这样：



Tips 鸟哥在接触 Linux 之前曾经碰过 Solaris 系统 （1999 年），当时鸟哥啥也不清楚！由于“听说”Linux 上面的帐号越复杂会导致系统越危险！所以鸟哥就将 /etc/passwd 上面的帐号全部删除到只剩下 root 与鸟哥自己用的一般帐号！结果你猜发生什么事？那就是....调用升阳的工程师来维护系统 @_@！糗到一个不行！大家不要学啊！

```
[root@study ~]# head -n 4 /etc/passwd
root:x:0:0:root:/root:/bin/bash  &lt;==等一下做为下面说明用
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
```

我们先来看一下每个 Linux 系统都会有的第一行，就是 root 这个系统管理员那一行好了， 你可以明显的看出来，每一行使用“:”分隔开，共有七个咚咚，分别是：

1. 帐号名称： 就是帐号啦！用来提供给对数字不太敏感的人类使用来登陆系统的！需要用来对应 UID 喔。例如 root 的 UID 对应就是 0 （第三字段）；

2. 密码： 早期 Unix 系统的密码就是放在这字段上！但是因为这个文件的特性是所有的程序都能够读取，这样一来很容易造成密码数据被窃取， 因此后来就将这个字段的密码数据给他改放到 /etc/shadow 中了。所以这里你会看到一个“ x ”，呵呵！

3. UID： 这个就是使用者识别码啰！通常 Linux 对于 UID 有几个限制需要说给您了解一下：

```
| id 范围 | 该 ID 使用者特性 |
| 0（系统管理员） | 当 UID 是 0 时，代表这个帐号是“系统管理员”！ 所以当你要让其他的帐号名称也具有
| 1~999（系统帐号） | 保留给系统使用的 ID，其实除了 0 之外，其他的 UID 权限与特性并没有不一样。默
| 1000~60000（可登陆帐号） | 给一般使用者用的。事实上，目前的 linux 核心 （3.10.x 版）已经可以
```

上面这样说明可以了解了吗？是的， UID 为 0 的时候，就是 root 呦！所以请特别留意一下你的 /etc/passwd 文件！

4. GID： 这个与 /etc/group 有关！其实 /etc/group 的观念与 /etc/passwd 差不多，只是他是用来规范群组名称与 GID 的对应而已！

5. 使用者信息说明栏： 这个字段基本上并没有什么重要用途，只是用来解释这个帐号的意义而已！不过，如果您提供使用 finger 的功能时， 这个字段可以提供很多的讯息呢！本章后面的 chfn 指令会来解释这里的说明。

6. 主文件夹： 这是使用者的主文件夹，以上面为例， root 的主文件夹在 /root ，所以当 root 登陆之后，就会立刻跑到 /root 目录里头啦！呵呵！ 如果你有个帐号的使用空间特别的大，你想要将该帐号的主文件夹移动到其他的硬盘去该怎么作？ 没有错！可以在这个字段进行修改呦！默认的使用者主文件夹在 /home/yourIDname

7. Shell： 我们在第十章 BASH 提到很多次，当使用者登陆系统后就会取得一个 Shell 来与系统的核心沟通以进行使用者的操作任务。那为何默认 shell 会使用 bash 呢？就是在这个字段指定的啰！ 这里比较需要注意的是，有一个 shell 可以用来替代成让帐号无法取得 shell 环境的登陆动作！那就是 /sbin/nologin 这个东西！这也可以用来制作纯 pop 邮件帐号者的数据呢！

8. /etc/shadow 文件结构

我们知道很多程序的运行都与权限有关，而权限与 UID/GID 有关！因此各程序当然需要读取 /etc/passwd 来了解不同帐号的权限。 因此 /etc/passwd 的权限需设置为 -rw-r--r-- 这样的情况， 虽然早期的密码也有加密过，但却放置到 /etc/passwd 的第二个字段上！这样一来很容易被有心人士所窃取的， 加密过的密码也能够通过暴力破解法去 trial and error （试误） 找出来！

因为这样的关系，所以后来发展出将密码移动到 /etc/shadow 这个文件分隔开来的技术， 而且还加入很多的密码限制参数在 /etc/shadow 里头呢！在这里，我们先来了解一下这个文件的构造吧！ 鸟哥的 /etc/shadow 文件有点像这样：

```
[root@study ~]# head -n 4 /etc/shadow
root:$6$wtbCCce/PxMeE5wm$KE2IfSJr.YLP7Rcai6oa/T7KFhO...:16559:0:99999:7:::  &lt;==下面说明
bin:*:16372:0:99999:7:::
daemon:*:16372:0:99999:7:::
adm:*:16372:0:99999:7:::
```

基本上， shadow 同样以“:”作为分隔符号，如果数一数，会发现共有九个字段啊，这九个字段的用途是这样的：

1. 帐号名称： 由于密码也需要与帐号对应啊～因此，这个文件的第一栏就是帐号，必须要与 /etc/passwd 相同才行！

2. 密码： 这个字段内的数据才是真正的密码，而且是经过编码的密码 （加密） 啦！ 你只会看到有一些特殊符号的字母就是了！需要特别留意的是，虽然这些加密过的密码很难被解出来， 但是“很难”不等于“不会”，所以，这个文件的默认权限是“-rw-------”或者是“----------”，亦即只有 root 才可以读写就是了！你得随时注意，不要不小心更动了这个文件的权限呢！

   另外，由于各种密码编码的技术不一样，因此不同的编码系统会造成这个字段的长度不相同。 举例来说，旧式的 DES, MD5 编码系统产生的密码长度就与目前惯用的 SHA 不同[2]！SHA 的密码长度明显的比较长些。由于固定的编码系统产生的密码长度必须一致，因此“当你让这个字段的长度改变后，该密码就会失效（算不出来）”。 很多软件通过这个功能，在此字段前加上 ! 或 * 改变密码字段长度，就会让密码“暂时失效”了。

3. 最近更动密码的日期： 这个字段记录了“更动密码那一天”的日期，不过，很奇怪呀！在我的例子中怎么会是 16559 呢？呵呵，这个是因为计算 Linux 日期的时间是以 1970 年 1 月 1 日作为 1 而累加的日期，1971 年 1 月 1 日则为 366 啦！ 得注意一下这个数据呦！上述的 16559 指的就是 2015-05-04 那一天啦！了解乎？ 而想要了解该日期可以使用本章后面 chage 指令的帮忙！至于想要知道某个日期的累积日数， 可使用如下的程序计算：

   ```
   [root@study ~]# echo $（（$（date --date="2015/05/04" +%s）/86400+1））
   16559
   ```

   上述指令中，2015/05/04 为你想要计算的日期，86400 为每一天的秒数， %s 为 1970/01/01 以来的累积总秒数。 由于 bash 仅支持整数，因此最终需要加上 1 补齐 1970/01/01 当天。

4. 密码不可被更动的天数：（与第 3 字段相比） 第四个字段记录了：这个帐号的密码在最近一次被更改后需要经过几天才可以再被变更！如果是 0 的话， 表示密码随时可以更动的意思。这的限制是为了怕密码被某些人一改再改而设计的！如果设置为 20 天的话，那么当你设置了密码之后， 20 天之内都无法改变这个密码呦！

5. 密码需要重新变更的天数：（与第 3 字段相比） 经常变更密码是个好习惯！为了强制要求使用者变更密码，这个字段可以指定在最近一次更改密码后， 在多少天数内需要再次的变更密码才行。你必须要在这个天数内重新设置你的密码，否则这个帐号的密码将会“变为过期特性”。 而如果像上面的 99999 （计算为 273 年） 的话，那就表示，呵呵，密码的变更没有强制性之意。

6. 密码需要变更期限前的警告天数：（与第 5 字段相比） 当帐号的密码有效期限快要到的时候 （第 5 字段），系统会依据这个字段的设置，发出“警告”言论给这个帐号，提醒他“再过 n 天你的密码就要过期了，请尽快重新设置你的密码呦！”，如上面的例子，则是密码到期之前的 7 天之内，系统会警告该用户。

7. 密码过期后的帐号宽限时间（密码失效日）：（与第 5 字段相比） 密码有效日期为“更新日期（第3字段）”+“重新变更日期（第5字段）”，过了该期限后使用者依旧没有更新密码，那该密码就算过期了。 虽然密码过期但是该帐号还是可以用来进行其他工作的，包括登陆系统取得 bash 。不过如果密码过期了， 那当你登陆系统时，系统会强制要求你必须要重新设置密码才能登陆继续使用喔，这就是密码过期特性。

   那这个字段的功能是什么呢？是在密码过期几天后，如果使用者还是没有登陆更改密码，那么这个帐号的密码将会“失效”， 亦即该帐号再也无法使用该密码登陆了。要注意密码过期与密码失效并不相同。

8. 帐号失效日期： 这个日期跟第三个字段一样，都是使用 1970 年以来的总日数设置。这个字段表示： 这个帐号在此字段规定的日期之后，将无法再使用。 就是所谓的“帐号失效”，此时不论你的密码是否有过期，这个“帐号”都不能再被使用！ 这个字段会被使用通常应该是在“收费服务”的系统中，你可以规定一个日期让该帐号不能再使用啦！

9. 保留： 最后一个字段是保留的，看以后有没有新功能加入。

举个例子来说好了，假如我的 dmtsai 这个使用者的密码栏如下所示：

```
dmtsai:$6$M4IphgNP2TmlXaSS$B418YFroYxxmm....:16559:5:60:7:5:16679:
```

这表示什么呢？先要注意的是 16559 是 2015/05/04 。所以 dmtsai 这个使用者的密码相关意义是：

- 由于密码几乎仅能单向运算（由明码计算成为密码，无法由密码反推回明码），因此由上表的数据我们无法得知 dmstai 的实际密码明文 （第二个字段）；
- 此帐号最近一次更动密码的日期是 2015/05/04 （16559）；
- 能够再次修改密码的时间是 5 天以后，也就是 2015/05/09 以前 dmtsai 不能修改自己的密码；如果使用者还是尝试要更动自己的密码，系统就会出现这样的讯息：

  ```
  You must wait longer to change your password
  passwd: Authentication token manipulation error
  ```

画面中告诉我们：你必须要等待更久的时间才能够变更密码之意啦！

- 由于密码过期日期定义为 60 天后，亦即累积日数为： 16559+60=16619，经过计算得到此日数代表日期为 2015/07/03。 这表示："使用者必须要在 2015/05/09 （前 5 天不能改） 到 2015/07/03 之间的 60 天限制内去修改自己的密码，若 2015/07/03 之后还是没有变更密码时，该密码就宣告为过期"了！
- 警告日期设为 7 天，亦即是密码过期日前的 7 天，在本例中则代表 2015/06/26 ~ 2015/07/03 这七天。 如果使用者一直没有更改密码，那么在这 7 天中，只要 dmtsai 登陆系统就会发现如下的讯息：

```
Warning: your password will expire in 5 days
```

- 如果该帐号一直到 2015/07/03 都没有更改密码，那么密码就过期了。但是由于有 5 天的宽限天数， 因此 dmtsai 在 2015/07/08 前都还可以使用旧密码登陆主机。 不过登陆时会出现强制更改密码的情况，画面有点像下面这样：

```
You are required to change your password immediately （password aged）
WARNING: Your password has expired.
You must change your password now and login again!
Changing password for user dmtsai.
Changing password for dmtsai
（current） UNIX password:
```

你必须要输入一次旧密码以及两次新密码后，才能够开始使用系统的各项资源。如果你是在 2015/07/08 以后尝试以 dmtsai 登陆的话，那么就会出现如下的错误讯息且无法登陆，因为此时你的密码就失效去啦！

```
Your account has expired; please contact your system administrator
```

- 如果使用者在 2015/07/03 以前变更过密码，那么第 3 个字段的那个 16559 的天数就会跟着改变，因此， 所有的限制日期也会跟着相对变动喔！^_^
- 无论使用者如何动作，到了 16679 （大约是 2015/09/01 左右） 该帐号就失效了～

通过这样的说明，您应该会比较容易理解了吧？由于 shadow 有这样的重要性，因此可不能随意修改喔！ 但在某些情况下面你得要使用各种方法来处理这个文件的！举例来说，常常听到人家说："我的密码忘记了"， 或者是"我的密码不晓得被谁改过，跟原先的不一样了"，这个时候怎么办？

- 一般用户的密码忘记了：这个最容易解决，请系统管理员帮忙， 他会重新设置好你的密码而不需要知道你的旧密码！利用 root 的身份使用 passwd 指令来处理即可。

- root 密码忘记了：这就麻烦了！因为你无法使用 root 的身份登陆了嘛！ 但我们知道 root 的密码在 /etc/shadow 当中，因此你可以使用各种可行的方法开机进入 Linux 再去修改。例如重新开机进入单人维护模式（第十九章）后，系统会主动的给予 root 权限的 bash 接口， 此时再以 passwd 修改密码即可；或以 Live CD 开机后挂载根目录去修改 /etc/shadow，将里面的 root 的密码字段清空， 再重新开机后 root 将不用密码即可登陆！登陆后再赶快以 passwd 指令去设置 root 密码即可。



Tips 曾经听过一则笑话，某位老师主要是在教授 Linux 操作系统，但是他是兼任的老师，因此对于该系的计算机环境不熟。 由于当初安装该计算机教室 Linux 操作系统的人员已经离职且找不到联络方式了，也就是说 root 密码已经没有人晓得了！ 此时该老师就对学生说：“在 Linux 里面 root 密码不见了，我们只能重新安装”...感觉有点无力～ 又是个被 Windows 制约的人才！

另外，由于 Linux 的新旧版本差异颇大，旧的版本 （CentOS 5.x 以前） 还活在很多服务器内！因此，如果你想要知道 shadow 是使用哪种加密的机制时， 可以通过下面的方法去查询喔！

```
[root@study ~]# authconfig --test &#124; grep hashing
 password hashing algorithm is sha512
# 这就是目前的密码加密机制！
```

13.1.3 关于群组： 有效与初始群组、groups, newgrp

认识了帐号相关的两个文件 /etc/passwd 与 /etc/shadow 之后，你或许还是会觉得奇怪， 那么群组的配置文件在哪里？还有，在 /etc/passwd 的第四栏不是所谓的 GID 吗？那又是啥？呵呵～此时就需要了解 /etc/group 与 /etc/gshadow 啰～

- /etc/group 文件结构

这个文件就是在记录 GID 与群组名称的对应了～鸟哥测试机的 /etc/group 内容有点像这样：

```
[root@study ~]# head -n 4 /etc/group
root:x:0:
bin:x:1:
daemon:x:2:
sys:x:3:
```

这个文件每一行代表一个群组，也是以冒号“:”作为字段的分隔符号，共分为四栏，每一字段的意义是：

1. 群组名称： 就是群组名称啦！同样用来给人类使用的，基本上需要与第三字段的 GID 对应。

2. 群组密码： 通常不需要设置，这个设置通常是给“群组管理员”使用的，目前很少有这个机会设置群组管理员啦！ 同样的，密码已经移动到 /etc/gshadow 去，因此这个字段只会存在一个“x”而已；

3. GID： 就是群组的 ID 啊。我们 /etc/passwd 第四个字段使用的 GID 对应的群组名，就是由这里对应出来的！

4. 此群组支持的帐号名称： 我们知道一个帐号可以加入多个群组，那某个帐号想要加入此群组时，将该帐号填入这个字段即可。 举例来说，如果我想要让 dmtsai 与 alex 也加入 root 这个群组，那么在第一行的最后面加上“dmtsai,alex”，注意不要有空格， 使成为“ root:x:0:dmtsai,alex ”就可以啰～

谈完了 /etc/passwd, /etc/shadow, /etc/group 之后，我们可以使用一个简单的图示来了解一下 UID / GID 与密码之间的关系， 图示如下。其实重点是 /etc/passwd 啦，其他相关的数据都是根据这个文件的字段去找寻出来的。 下图中， root 的 UID 是 0 ，而 GID 也是 0 ，去找 /etc/group 可以知道 GID 为 0 时的群组名称就是 root 哩。 至于密码的寻找中，会找到 /etc/shadow 与 /etc/passwd 内同帐号名称的那一行，就是密码相关数据啰。

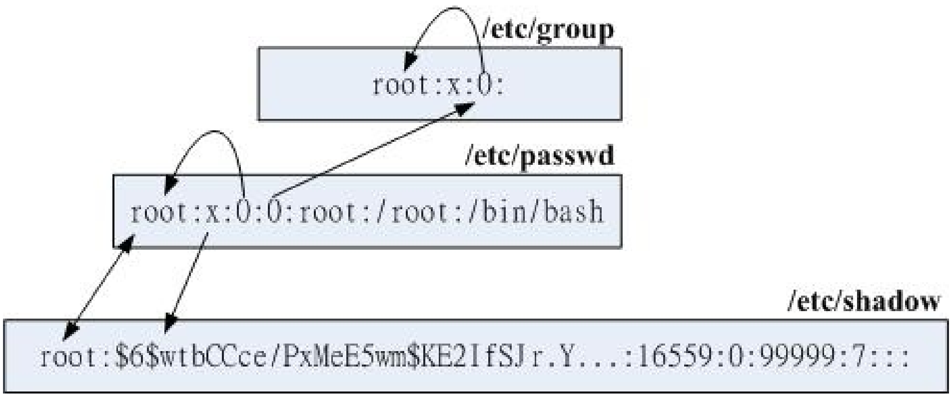


图13.1.1、帐号相关文件之间的 UID/GID 与密码相关性示意图

至于在 /etc/group 比较重要的特色在于第四栏啦，因为每个使用者都可以拥有多个支持的群组，这就好比在学校念书的时候， 我们可以加入多个社团一样！ ^_^。不过这里你或许会觉得奇怪的，那就是：“假如我同时加入多个群组，那么我在作业的时候，到底是以那个群组为准？” 下面我们就来谈一谈这个“有效群组”的概念。



Tips 请注意，新版的 Linux 中，初始群组的用户群已经不会加入在第四个字段！例如我们知道 root 这个帐号的主要群组为 root，但是在上面的范例中， 你已经不会看到 root 这个“用户”的名称在 /etc/group 的 root 那一行的第四个字段内啰！这点还请留意一下即可！

- 有效群组（effective group）与初始群组（initial group）

还记得每个使用者在他的 /etc/passwd 里面的第四栏有所谓的 GID 吧？那个 GID 就是所谓的“初始群组 （initial group） ”！也就是说，当使用者一登陆系统，立刻就拥有这个群组的相关权限的意思。 举例来说，我们上面提到 dmtsai 这个使用者的 /etc/passwd 与 /etc/group 还有 /etc/gshadow 相关的内容如下：

```
[root@study ~]# usermod -a -G users dmtsai  &lt;==先设置好次要群组
[root@study ~]# grep dmtsai /etc/passwd /etc/group /etc/gshadow
/etc/passwd:dmtsai:x:1000:1000:dmtsai:/home/dmtsai:/bin/bash
/etc/group:wheel:x:10:dmtsai    &lt;==次要群组的设置、安装时指定的
/etc/group:users:x:100:dmtsai   &lt;==次要群组的设置
/etc/group:dmtsai:x:1000:       &lt;==因为是初始群组，所以第四字段不需要填入帐号
/etc/gshadow:wheel:::dmtsai     &lt;==次要群组的设置
/etc/gshadow:users:::dmtsai     &lt;==次要群组的设置
/etc/gshadow:dmtsai:!!::
```

仔细看到上面这个表格，在 /etc/passwd 里面，dmtsai 这个使用者所属的群组为 GID=1000 ，搜寻一下 /etc/group 得到 1000 是那个名为 dmtsai 的群组啦！这就是 initial group。因为是初始群组， 使用者一登陆就会主动取得，不需要在 /etc/group 的第四个字段写入该帐号的！

但是非 initial group 的其他群组可就不同了。举上面这个例子来说，我将 dmtsai 加入 users 这个群组当中，由于 users 这个群组并非是 dmtsai 的初始群组，因此， 我必须要在 /etc/group 这个文件中，找到 users 那一行，并且将 dmtsai 这个帐号加入第四栏， 这样 dmtsai 才能够加入 users 这个群组啊。

那么在这个例子当中，因为我的 dmtsai 帐号同时支持 dmtsai, wheel 与 users 这三个群组，因此，在读取/写入/可执行文件案时，针对群组部分，只要是 users, wheel 与 dmtsai 这三个群组拥有的功能， 我 dmtsai 这个使用者都能够拥有喔！这样瞭呼？不过，这是针对已经存在的文件而言， 如果今天我要创建一个新的文件或者是新的目录，请问一下，新文件的群组是 dmtsai, wheel 还是 users ？呵呵！这就得要检查一下当时的有效群组了 （effective group）。

- groups: 有效与支持群组的观察

如果我以 dmtsai 这个使用者的身份登陆后，该如何知道我所有支持的群组呢？ 很简单啊，直接输入 groups 就可以了！注意喔，是 groups 有加 s 呢！结果像这样：

```
[dmtsai@study ~]$ groups
dmtsai wheel users
```

在这个输出的讯息中，可知道 dmtsai 这个用户同时属于 dmtsai, wheel 及 users 这三个群组，而且， 第一个输出的群组即为有效群组 （effective group） 了。 也就是说，我的有效群组为 dmtsai 啦～此时，如果我以 touch 去创建一个新文件，例如：" touch test "，那么这个文件的拥有者为 dmtsai ，而且群组也是 dmtsai 的啦。

```
[dmtsai@study ~]$ touch test
[dmtsai@study ~]$ ll test
-rw-rw-r--. 1 dmtsai dmtsai 0 Jul 20 19:54 test
```

这样是否可以了解什么是有效群组了？通常有效群组的作用是在新建文件啦！那么有效群组是否能够变换？

- newgrp: 有效群组的切换

那么如何变更有效群组呢？就使用 newgrp 啊！不过使用 newgrp 是有限制的，那就是你想要切换的群组必须是你已经有支持的群组。举例来说， dmtsai 可以在 dmtsai/wheel/users 这三个群组间切换有效群组，但是 dmtsai 无法切换有效群组成为 sshd 啦！使用的方式如下：

```
[dmtsai@study ~]$ newgrp users
[dmtsai@study ~]$ groups
users wheel dmtsai
[dmtsai@study ~]$ touch test2
[dmtsai@study ~]$ ll test*
-rw-rw-r--. 1 dmtsai dmtsai 0 Jul 20 19:54 test
-rw-r--r--. 1 dmtsai users  0 Jul 20 19:56 test2
[dmtsai@study ~]$ exit   # 注意！记得离开 newgrp 的环境喔！
```

此时，dmtsai 的有效群组就成为 users 了。我们额外的来讨论一下 newgrp 这个指令，这个指令可以变更目前使用者的有效群组， 而且是另外以一个 shell 来提供这个功能的喔，所以，以上面的例子来说， dmtsai 这个使用者目前是以另一个 shell 登陆的，而且新的 shell 给予 dmtsai 有效 GID 为 users 就是了。如果以图示来看就是如下所示：

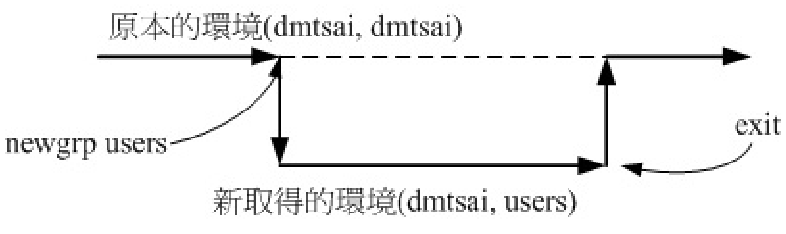


图13.1.2、newgrp 的运行示意图

虽然使用者的环境设置（例如环境变量等等其他数据）不会有影响，但是使用者的"群组权限"将会重新被计算。 但是需要注意，由于是新取得一个 shell ，因此如果你想要回到原本的环境中，请输入 exit 回到原本的 shell 喔！

既然如此，也就是说，只要我的用户有支持的群组就是能够切换成为有效群组！好了， 那么如何让一个帐号加入不同的群组就是问题的所在啰。你要加入一个群组有两个方式，一个是通过系统管理员 （root） 利用 usermod 帮你加入，如果 root 太忙了而且你的系统有设置群组管理员，那么你可以通过群组管理员以 gpasswd 帮你加入他所管理的群组中！详细的作法留待下一小节再来介绍啰！

- /etc/gshadow

刚刚讲了很多关于“有效群组”的概念，另外，也提到 newgrp 这个指令的用法，但是，如果 /etc/gshadow 这个设置没有搞懂得话，那么 newgrp 是无法动作的呢！ 鸟哥测试机的 /etc/gshadow 的内容有点像这样：

```
[root@study ~]# head -n 4 /etc/gshadow
root:::
bin:::
daemon:::
sys:::
```

这个文件内同样还是使用冒号“:”来作为字段的分隔字符，而且你会发现，这个文件几乎与 /etc/group 一模一样啊！是这样没错～不过，要注意的大概就是第二个字段吧～第二个字段是密码栏， 如果密码栏上面是“!”或空的时，表示该群组不具有群组管理员！至于第四个字段也就是支持的帐号名称啰～ 这四个字段的意义为：

1. 群组名称
2. 密码栏，同样的，开头为！表示无合法密码，所以无群组管理员
3. 群组管理员的帐号 （相关信息在 gpasswd 中介绍）
4. 有加入该群组支持的所属帐号 （与 /etc/group 内容相同！）

以系统管理员的角度来说，这个 gshadow 最大的功能就是创建群组管理员啦！ 那么什么是群组管理员呢？由于系统上面的帐号可能会很多，但是我们 root 可能平时太忙碌，所以当有使用者想要加入某些群组时， root 或许会没有空管理。此时如果能够创建群组管理员的话，那么该群组管理员就能够将那个帐号加入自己管理的群组中！ 可以免去 root 的忙碌啦！不过，由于目前有类似 sudo 之类的工具， 所以这个群组管理员的功能已经很少使用了。我们会在后续的 gpasswd 中介绍这个实作。

13.2 帐号管理

好啦！既然要管理帐号，当然是由新增与移除使用者开始的啰～下面我们就分别来谈一谈如何新增、 移除与更改使用者的相关信息吧～

13.2.1 新增与移除使用者： useradd, 相关配置文件, passwd, usermod, userdel

要如何在 Linux 的系统新增一个使用者啊？呵呵～真是太简单了～我们登陆系统时会输入（1）帐号与（2）密码，所以创建一个可用的帐号同样的也需要这两个数据。那帐号可以使用 useradd 来新建使用者，密码的给予则使用 passwd 这个指令！这两个指令下达方法如下：

- useradd

```
[root@study ~]# useradd [-u UID] [-g 初始群组] [-G 次要群组] [-mM]\
&gt;  [-c 说明栏] [-d 主文件夹绝对路径] [-s shell] 使用者帐号名
选项与参数：
-u  ：后面接的是 UID ，是一组数字。直接指定一个特定的 UID 给这个帐号；
-g  ：后面接的那个群组名称就是我们上面提到的 initial group 啦～
      该群组的 GID 会被放置到 /etc/passwd 的第四个字段内。
-G  ：后面接的群组名称则是这个帐号还可以加入的群组。
      这个选项与参数会修改 /etc/group 内的相关数据喔！
-M  ：强制！不要创建使用者主文件夹！（系统帐号默认值）
-m  ：强制！要创建使用者主文件夹！（一般帐号默认值）
-c  ：这个就是 /etc/passwd 的第五栏的说明内容啦～可以随便我们设置的啦～
-d  ：指定某个目录成为主文件夹，而不要使用默认值。务必使用绝对路径！
-r  ：创建一个系统的帐号，这个帐号的 UID 会有限制 （参考 /etc/login.defs）
-s  ：后面接一个 shell ，若没有指定则默认是 /bin/bash 的啦～
-e  ：后面接一个日期，格式为“YYYY-MM-DD”此项目可写入 shadow 第八字段，
      亦即帐号失效日的设置项目啰；
-f  ：后面接 shadow 的第七字段项目，指定密码是否会失效。0为立刻失效，
      -1 为永远不失效（密码只会过期而强制于登陆时重新设置而已。）

范例一：完全参考默认值创建一个使用者，名称为 vbird1
[root@study ~]# useradd vbird1
[root@study ~]# ll -d /home/vbird1
drwx------. 3 vbird1 vbird1 74 Jul 20 21:50 /home/vbird1
# 默认会创建使用者主文件夹，且权限为 700 ！这是重点！

[root@study ~]# grep vbird1 /etc/passwd /etc/shadow /etc/group
/etc/passwd:vbird1:x:1003:1004::/home/vbird1:/bin/bash
/etc/shadow:vbird1:!!:16636:0:99999:7:::
/etc/group:vbird1:x:1004:     &lt;==默认会创建一个与帐号一模一样的群组名
```

其实系统已经帮我们规定好非常多的默认值了，所以我们可以简单的使用“ useradd 帐号 ”来创建使用者即可。 CentOS 这些默认值主要会帮我们处理几个项目：

- 在 /etc/passwd 里面创建一行与帐号相关的数据，包括创建 UID/GID/主文件夹等；
- 在 /etc/shadow 里面将此帐号的密码相关参数填入，但是尚未有密码；
- 在 /etc/group 里面加入一个与帐号名称一模一样的群组名称；
- 在 /home 下面创建一个与帐号同名的目录作为使用者主文件夹，且权限为 700

由于在 /etc/shadow 内仅会有密码参数而不会有加密过的密码数据，因此我们在创建使用者帐号时， 还需要使用“ passwd 帐号 ”来给予密码才算是完成了使用者创建的流程。如果由于特殊需求而需要改变使用者相关参数时， 就得要通过上述表格中的选项来进行创建了，参考下面的案例：

```
范例二：假设我已知道我的系统当中有个群组名称为 users ，且 UID 1500 并不存在，
        请用 users 为初始群组，以及 uid 为 1500 来创建一个名为 vbird2 的帐号
[root@study ~]# useradd -u 1500 -g users vbird2
[root@study ~]# ll -d /home/vbird2
drwx------. 3 vbird2 users 74 Jul 20 21:52 /home/vbird2

[root@study ~]# grep vbird2 /etc/passwd /etc/shadow /etc/group
/etc/passwd:vbird2:x:1500:100::/home/vbird2:/bin/bash
/etc/shadow:vbird2:!!:16636:0:99999:7:::
# 看一下，UID 与 initial group 确实改变成我们需要的了！
```

在这个范例中，我们创建的是指定一个已经存在的群组作为使用者的初始群组，因为群组已经存在， 所以在 /etc/group 里面就不会主动的创建与帐号同名的群组了！ 此外，我们也指定了特殊的 UID 来作为使用者的专属 UID 喔！了解了一般帐号后，我们来瞧瞧那啥是系统帐号（system account） 吧！

```
范例三：创建一个系统帐号，名称为 vbird3
[root@study ~]# useradd -r vbird3
[root@study ~]# ll -d /home/vbird3
ls: cannot access /home/vbird3: No such file or directorya   &lt;==不会主动创建主文件夹

[root@study ~]# grep vbird3 /etc/passwd /etc/shadow /etc/group
/etc/passwd:vbird3:x:699:699::/home/vbird3:/bin/bash
/etc/shadow:vbird3:!!:16636::::::
/etc/group:vbird3:x:699:
```

我们在谈到 UID 的时候曾经说过一般帐号应该是 1000 号以后，那使用者自己创建的系统帐号则一般是小于 1000 号以下的。 所以在这里我们加上 -r 这个选项以后，系统就会主动将帐号与帐号同名群组的 UID/GID 都指定小于 1000 以下， 在本案例中则是使用 699（UID） 与 699（GID） 啰！此外，由于系统帐号主要是用来进行运行系统所需服务的权限设置， 所以系统帐号默认都不会主动创建主文件夹的！

由这几个范例我们也会知道，使用 useradd 创建使用者帐号时，其实会更改不少地方，至少我们就知道下面几个文件：

- 使用者帐号与密码参数方面的文件：/etc/passwd, /etc/shadow
- 使用者群组相关方面的文件：/etc/group, /etc/gshadow
- 使用者的主文件夹：/home/帐号名称

那请教一下，你有没有想过，为何“ useradd vbird1 ”会主动在 /home/vbird1 创建起使用者的主文件夹？主文件夹内有什么数据且来自哪里？为何默认使用的是 /bin/bash 这个 shell ？为何密码字段已经都规范好了 （0:99999:7 那一串）？呵呵！这就得要说明一下 useradd 所使用的参考文件啰！

- useradd 参考档

其实 useradd 的默认值可以使用下面的方法调用出来：

```
[root@study ~]# useradd -D
GROUP=100         &lt;==默认的群组
HOME=/home         &lt;==默认的主文件夹所在目录
INACTIVE=-1         &lt;==密码失效日，在 shadow 内的第 7 栏
EXPIRE=             &lt;==帐号失效日，在 shadow 内的第 8 栏
SHELL=/bin/bash         &lt;==默认的 shell
SKEL=/etc/skel         &lt;==使用者主文件夹的内容数据参考目录
CREATE_MAIL_SPOOL=yes    &lt;==是否主动帮使用者创建邮件信箱（mailbox）
```

这个数据其实是由 /etc/default/useradd 调用出来的！你可以自行用 vim 去观察该文件的内容。搭配上头刚刚谈过的范例一的运行结果，上面这些设置项目所造成的行为分别是：

- GROUP=100：新建帐号的初始群组使用 GID 为 100 者

系统上面 GID 为 100 者即是 users 这个群组，此设置项目指的就是让新设使用者帐号的初始群组为 users 这一个的意思。 但是我们知道 CentOS 上面并不是这样的，在 CentOS 上面默认的群组为与帐号名相同的群组。 举例来说， vbird1 的初始群组为 vbird1 。怎么会这样啊？这是因为针对群组的角度有两种不同的机制所致， 这两种机制分别是：

- 私有群组机制：

系统会创建一个与帐号一样的群组给使用者作为初始群组。 这种群组的设置机制会比较有保密性，这是因为使用者都有自己的群组，而且主文件夹权限将会设置为 700 （仅有自己可进入自己的主文件夹） 之故。使用这种机制将不会参考 GROUP=100 这个设置值。代表性的 distributions 有 RHEL, Fedora, CentOS 等；

- 公共群组机制：

就是以 GROUP=100 这个设置值作为新建帐号的初始群组，因此每个帐号都属于 users 这个群组， 且默认主文件夹通常的权限会是“ drwxr-xr-x ... username users ... ”，由于每个帐号都属于 users 群组，因此大家都可以互相分享主文件夹内的数据之故。代表 distributions 如 SuSE等。

由于我们的 CentOS 使用私有群组机制，因此这个设置项目是不会生效的！不要太紧张啊！

- HOME=/home：使用者主文件夹的基准目录（basedir）

使用者的主文件夹通常是与帐号同名的目录，这个目录将会摆放在此设置值的目录后。所以 vbird1 的主文件夹就会在 /home/vbird1/ 了！很容易理解吧！

- INACTIVE=-1：密码过期后是否会失效的设置值

我们在 shadow 文件结构当中谈过，第七个字段的设置值将会影响到密码过期后， 在多久时间内还可使用旧密码登陆。这个项目就是在指定该日数啦！如果是 0 代表密码过期立刻失效， 如果是 -1 则是代表密码永远不会失效，如果是数字，如 30 ，则代表过期 30 天后才失效。

- EXPIRE=：帐号失效的日期

就是 shadow 内的第八字段，你可以直接设置帐号在哪个日期后就直接失效，而不理会密码的问题。 通常不会设置此项目，但如果是付费的会员制系统，或许这个字段可以设置喔！

- SHELL=/bin/bash：默认使用的 shell 程序文件名

系统默认的 shell 就写在这里。假如你的系统为 mail server ，你希望每个帐号都只能使用 email 的收发信件功能， 而不许使用者登陆系统取得 shell ，那么可以将这里设置为 /sbin/nologin ，如此一来，新建的使用者默认就无法登陆！ 也免去后续使用 usermod 进行修改的手续！

- SKEL=/etc/skel：使用者主文件夹参考基准目录

这个咚咚就是指定使用者主文件夹的参考基准目录啰～举我们的范例一为例， vbird1 主文件夹 /home/vbird1 内的各项数据，都是由 /etc/skel 所复制过去的～所以呢，未来如果我想要让新增使用者时，该使用者的环境变量 ~/.bashrc 就设置妥当的话，您可以到 /etc/skel/.bashrc 去编辑一下，也可以创建 /etc/skel/www 这个目录，那么未来新增使用者后，在他的主文件夹下就会有 www 那个目录了！这样瞭呼？

- CREATE_MAIL_SPOOL=yes：创建使用者的 mailbox

你可以使用“ ll /var/spool/mail/vbird1 ”看一下，会发现有这个文件的存在喔！这就是使用者的邮件信箱！

除了这些基本的帐号设置值之外， UID/GID 还有密码参数又是在哪里参考的呢？那就得要看一下 /etc/login.defs 啦！ 这个文件的内容有点像下面这样：

```
MAIL_DIR        /var/spool/mail  &lt;==使用者默认邮件信箱放置目录

PASS_MAX_DAYS   99999    &lt;==/etc/shadow 内的第 5 栏，多久需变更密码日数
PASS_MIN_DAYS   0        &lt;==/etc/shadow 内的第 4 栏，多久不可重新设置密码日数
PASS_MIN_LEN    5        &lt;==密码最短的字符长度，已被 pam 模块取代，失去效用！
PASS_WARN_AGE   7        &lt;==/etc/shadow 内的第 6 栏，过期前会警告的日数

UID_MIN          1000    &lt;==使用者最小的 UID，意即小于 1000 的 UID 为系统保留
UID_MAX         60000    &lt;==使用者能够用的最大 UID
SYS_UID_MIN       201    &lt;==保留给使用者自行设置的系统帐号最小值 UID
SYS_UID_MAX       999    &lt;==保留给使用者自行设置的系统帐号最大值 UID
GID_MIN          1000    &lt;==使用者自订群组的最小 GID，小于 1000 为系统保留
GID_MAX         60000    &lt;==使用者自订群组的最大 GID
SYS_GID_MIN       201    &lt;==保留给使用者自行设置的系统帐号最小值 GID
SYS_GID_MAX       999    &lt;==保留给使用者自行设置的系统帐号最大值 GID

CREATE_HOME     yes      &lt;==在不加 -M 及 -m 时，是否主动创建使用者主文件夹？
UMASK           077      &lt;==使用者主文件夹创建的 umask ，因此权限会是 700
USERGROUPS_ENAB yes      &lt;==使用 userdel 删除时，是否会删除初始群组
ENCRYPT_METHOD SHA512    &lt;==密码加密的机制使用的是 sha512 这一个机制！
```

这个文件规范的数据则是如下所示：

- mailbox 所在目录： 使用者的默认 mailbox 文件放置的目录在 /var/spool/mail，所以 vbird1 的 mailbox 就是在 /var/spool/mail/vbird1 啰！

- shadow 密码第 4, 5, 6 字段内容： 通过 PASS*MAX_DAYS* 等等设置值来指定的！所以你知道为何默认的 */etc/shadow* 内每一行都会有“ *0:99999:7* ”的存在了吗？^^！不过要注意的是，由于目前我们登陆时改用 PAM 模块来进行密码检验，所以那个 PASS_MIN_LEN 是失效的！

- UID/GID 指定数值： 虽然 Linux 核心支持的帐号可高达 232 这么多个，不过一部主机要作出这么多帐号在管理上也是很麻烦的！ 所以在这里就针对 UID/GID 的范围进行规范就是了。上表中的 UID_MIN 指的就是可登陆系统的一般帐号的最小 UID ，至于 UID_MAX 则是最大 UID 之意。

  要注意的是，系统给予一个帐号 UID 时，他是 （1）先参考 UID_MIN 设置值取得最小数值； （2）由 /etc/passwd 搜寻最大的 UID 数值， 将 （1） 与 （2） 相比，找出最大的那个再加一就是新帐号的 UID 了。我们上面已经作出 UID 为 1500 的 vbird2 ， 如果再使用“ useradd vbird4 ”时，你猜 vbird4 的 UID 会是多少？答案是： 1501 。 所以中间的 1004~1499 的号码就空下来啦！

  而如果我是想要创建系统用的帐号，所以使用 useradd -r sysaccount 这个 -r 的选项时，就会找“比 201 大但比 1000 小的最大的 UID ”就是了。 ^_^

- 使用者主文件夹设置值： 为何我们系统默认会帮使用者创建主文件夹？就是这个“CREATE_HOME = yes”的设置值啦！这个设置值会让你在使用 useradd 时， 主动加入“ -m ”这个产生主文件夹的选项啊！如果不想要创建使用者主文件夹，就只能强制加上“ -M ”的选项在 useradd 指令执行时啦！至于创建主文件夹的权限设置呢？就通过 umask 这个设置值啊！因为是 077 的默认设置，因此使用者主文件夹默认权限才会是“ drwx------ ”哩！

- 使用者删除与密码设置值： 使用“USERGROUPS_ENAB yes”这个设置值的功能是： 如果使用 userdel 去删除一个帐号时，且该帐号所属的初始群组已经没有人隶属于该群组了， 那么就删除掉该群组，举例来说，我们刚刚有创建 vbird4 这个帐号，他会主动创建 vbird4 这个群组。 若 vbird4 这个群组并没有其他帐号将他加入支持的情况下，若使用 userdel vbird4 时，该群组也会被删除的意思。 至于“ENCRYPT_METHOD SHA512”则表示使用 SHA512 来加密密码明文，而不使用旧式的 MD5。

现在你知道啦，使用 useradd 这支程序在创建 Linux 上的帐号时，至少会参考：

- /etc/default/useradd
- /etc/login.defs
- /etc/skel/*

这些文件，不过，最重要的其实是创建 /etc/passwd, /etc/shadow, /etc/group, /etc/gshadow 还有使用者主文件夹就是了～所以，如果你了解整个系统运行的状态，也是可以手动直接修改这几个文件就是了。 OK！帐号创建了，接下来处理一下使用者的密码吧！

- passwd

刚刚我们讲到了，使用 useradd 创建了帐号之后，在默认的情况下，该帐号是暂时被封锁的， 也就是说，该帐号是无法登陆的，你可以去瞧一瞧 /etc/shadow 内的第二个字段就晓得啰～ 那该如何是好？怕什么？直接给他设置新密码就好了嘛！对吧～设置密码就使用 passwd 啰！

```
[root@study ~]# passwd [--stdin] [帐号名称]  &lt;==所有人均可使用来改自己的密码
[root@study ~]# passwd [-l] [-u] [--stdin] [-S] \
&gt;  [-n 日数] [-x 日数] [-w 日数] [-i 日期] 帐号 &lt;==root 功能
选项与参数：
--stdin ：可以通过来自前一个管线的数据，作为密码输入，对 shell script 有帮助！
-l  ：是 Lock 的意思，会将 /etc/shadow 第二栏最前面加上 ! 使密码失效；
-u  ：与 -l 相对，是 Unlock 的意思！
-S  ：列出密码相关参数，亦即 shadow 文件内的大部分信息。
-n  ：后面接天数，shadow 的第 4 字段，多久不可修改密码天数
-x  ：后面接天数，shadow 的第 5 字段，多久内必须要更动密码
-w  ：后面接天数，shadow 的第 6 字段，密码过期前的警告天数
-i  ：后面接“日期”，shadow 的第 7 字段，密码失效日期

范例一：请 root 给予 vbird2 密码
[root@study ~]# passwd vbird2
Changing password for user vbird2.
New UNIX password: &lt;==这里直接输入新的密码，屏幕不会有任何反应
BAD PASSWORD: The password is shorter than 8 characters &lt;==密码太简单或过短的错误！
Retype new UNIX password:  &lt;==再输入一次同样的密码
passwd: all authentication tokens updated successfully.  &lt;==竟然还是成功修改了！
```

root 果然是最伟大的人物！当我们要给予使用者密码时，通过 root 来设置即可。 root 可以设置各式各样的密码，系统几乎一定会接受！所以您瞧瞧，如同上面的范例一，明明鸟哥输入的密码太短了， 但是系统依旧可接受 vbird2 这样的密码设置。这个是 root 帮忙设置的结果，那如果是使用者自己要改密码呢？ 包括 root 也是这样修改的喔！

```
范例二：用 vbird2 登陆后，修改 vbird2 自己的密码
[vbird2@study ~]$ passwd   &lt;==后面没有加帐号，就是改自己的密码！
Changing password for user vbird2.
Changing password for vbird2
（current） UNIX password: &lt;==这里输入“原有的旧密码”
New UNIX password: &lt;==这里输入新密码
BAD PASSWORD: The password is shorter than 8 characters &lt;==密码太短！不可以设置！重新想
New password:  &lt;==这里输入新想的密码
BAD PASSWORD: The password fails the dictionary check - it is based on a dictionary word
# 同样的，密码设置在字典里面找的到该字串，所以也是不建议！无法通过，再想新的！
New UNIX password: &lt;==这里再想个新的密码来输入吧
Retype new UNIX password: &lt;==通过密码验证！所以重复这个密码的输入
passwd: all authentication tokens updated successfully. &lt;==有无成功看关键字
```

passwd 的使用真的要很注意，尤其是 root 先生啊！鸟哥在课堂上每次讲到这里，说是要帮自己的一般帐号创建密码时， 有一小部分的学生就是会忘记加上帐号，结果就变成改变 root 自己的密码，最后.... root 密码就这样不见去！唉～ 要帮一般帐号创建密码需要使用“ passwd 帐号 ”的格式，使用“ passwd ”表示修改自己的密码！拜托！千万不要改错！

与 root 不同的是，一般帐号在更改密码时需要先输入自己的旧密码 （亦即 current 那一行），然后再输入新密码 （New 那一行）。 要注意的是，密码的规范是非常严格的，尤其新的 distributions 大多使用 PAM 模块来进行密码的检验，包括太短、 密码与帐号相同、密码为

字典常见字串等，都会被 PAM 模块检查出来而拒绝修改密码，此时会再重复出现“ New ”这个关键字！ 那时请再想个新密码！若出现“ Retype ”才是你的密码被接受了！重复输入新密码并且看到“ successfully ”这个关键字时才是修改密码成功喔！



Tips 与一般使用者不同的是， root 并不需要知道旧密码就能够帮使用者或 root 自己创建新密码！ 但如此一来有困扰～就是如果你的亲密爱人老是告诉你“我的密码真难记，帮我设置简单一点的！”时， 千万不要妥协啊！这是为了系统安全...

为何使用者要设订自己的密码会这么麻烦啊？这是因为密码的安全性啦！如果密码设置太简单， 一些有心人士就能够很简单的猜到你的密码，如此一来人家就可能使用你的一般帐号登陆你的主机或使用其他主机资源， 对主机的维护会造成困扰的！所以新的 distributions 是使用较严格的 PAM 模块来管理密码，这个管理的机制写在 /etc/pam.d/passwd 当中。而该文件与密码有关的测试模块就是使用：pam_cracklib.so，这个模块会检验密码相关的信息， 并且取代 /etc/login.defs 内的 PASS_MIN_LEN 的设置啦！关于 PAM 我们在本章后面继续介绍，这里先谈一下， 理论上，你的密码最好符合如下要求：

- 密码不能与帐号相同；
- 密码尽量不要选用字典里面会出现的字串；
- 密码需要超过 8 个字符；
- 密码不要使用个人信息，如身份证、手机号码、其他电话号码等；
- 密码不要使用简单的关系式，如 1+1=2， Iamvbird 等；
- 密码尽量使用大小写字符、数字、特殊字符（$,_,-等）的组合。

为了方便系统管理，新版的 passwd 还加入了很多创意选项喔！鸟哥个人认为最好用的大概就是这个“ --stdin ”了！ 举例来说，你想要帮 vbird2 变更密码成为 abc543CC ，可以这样下达指令呢！

```
范例三：使用 standard input 创建用户的密码
[root@study ~]# echo "abc543CC" &#124; passwd --stdin vbird2
Changing password for user vbird2.
passwd: all authentication tokens updated successfully.
```

这个动作会直接更新使用者的密码而不用再次的手动输入！好处是方便处理，缺点是这个密码会保留在指令中， 未来若系统被攻破，人家可以在 /root/.bash_history 找到这个密码呢！ 所以这个动作通常仅用在 shell script 的大量创建使用者帐号当中！要注意的是，这个选项并不存在所有 distributions 版本中， 请使用 man passwd 确认你的 distribution 是否有支持此选项喔！

如果你想要让 vbird2 的密码具有相当的规则，举例来说你要让 vbird2 每 60 天需要变更密码， 密码过期后 10 天未使用就宣告帐号失效，那该如何处理？

```
范例四：管理 vbird2 的密码使具有 60 天变更、密码过期 10 天后帐号失效的设置
[root@study ~]# passwd -S vbird2
vbird2 PS 2015-07-20 0 99999 7 -1 (Password set, SHA512 crypt.)
# 上面说明密码创建时间 (2015-07-20)、0 最小天数、99999 变更天数、7 警告日数与密码不会失效 (-1)

[root@study ~]# passwd -x 60 -i 10 vbird2
[root@study ~]# passwd -S vbird2
vbird2 PS 2015-07-20 0 60 7 10 (Password set, SHA512 crypt.)
```

那如果我想要让某个帐号暂时无法使用密码登陆主机呢？举例来说， vbird2 这家伙最近老是胡乱在主机乱来， 所以我想要暂时让她无法登陆的话，最简单的方法就是让她的密码变成不合法 (shadow 第 2 字段长度变掉)！ 处理的方法就更简单的！

```
范例五：让 vbird2 的帐号失效，观察完毕后再让她失效
[root@study ~]# passwd -l vbird2
[root@study ~]# passwd -S vbird2
vbird2 LK 2015-07-20 0 60 7 10 (Password locked.)
# 嘿嘿！状态变成“ LK, Lock ”了啦！无法登陆喔！
[root@study ~]# grep vbird2 /etc/shadow
vbird2:!!$6$iWWO6T46$uYStdkB7QjcUpJaCLB.OOp...:16636:0:60:7:10::
# 其实只是在这里加上 !! 而已！

[root@study ~]# passwd -u vbird2
[root@study ~]# grep vbird2 /etc/shadow
vbird2:$6$iWWO6T46$uYStdkB7QjcUpJaCLB.OOp...:16636:0:60:7:10::
# 密码字段恢复正常！
```

是否很有趣啊！您可以自行管理一下你的帐号的密码相关参数喔！接下来让我们用更简单的方法来查阅密码参数喔！

- chage

除了使用 passwd -S 之外，有没有更详细的密码参数显示功能呢？有的！那就是 chage 了！他的用法如下：

```
[root@study ~]# chage [-ldEImMW] 帐号名
选项与参数：
-l ：列出该帐号的详细密码参数；
-d ：后面接日期，修改 shadow 第三字段(最近一次更改密码的日期)，格式 YYYY-MM-DD
-E ：后面接日期，修改 shadow 第八字段(帐号失效日)，格式 YYYY-MM-DD
-I ：后面接天数，修改 shadow 第七字段(密码失效日期)
-m ：后面接天数，修改 shadow 第四字段(密码最短保留天数)
-M ：后面接天数，修改 shadow 第五字段(密码多久需要进行变更)
-W ：后面接天数，修改 shadow 第六字段(密码过期前警告日期)

范例一：列出 vbird2 的详细密码参数
[root@study ~]# chage -l vbird2
Last password change                                    : Jul 20, 2015
Password expires                                        : Sep 18, 2015
Password inactive                                       : Sep 28, 2015
Account expires                                         : never
Minimum number of days between password change          : 0
Maximum number of days between password change          : 60
Number of days of warning before password expires       : 7
```

我们在 passwd 的介绍中谈到了处理 vbird2 这个帐号的密码属性流程，使用 passwd -S 却无法看到很清楚的说明。如果使用 chage 那可就明白多了！如上表所示，我们可以清楚的知道 vbird2 的详细参数呢！ 如果想要修改其他的设置值，就自己参考上面的选项，或者自行 man chage 一下吧！^_^

chage 有一个功能很不错喔！如果你想要让“使用者在第一次登陆时， 强制她们一定要更改密码后才能够使用系统资源”，可以利用如下的方法来处理的！

```
范例二：创建一个名为 agetest 的帐号，该帐号第一次登陆后使用默认密码，但必须要更改过密码后，
        使用新密码才能够登陆系统使用 bash 环境
[root@study ~]# useradd agetest
[root@study ~]# echo "agetest" &#124; passwd --stdin agetest
[root@study ~]# chage -d 0 agetest
[root@study ~]# chage -l agetest &#124; head -n 3
Last password change                    : password must be changed
Password expires                        : password must be changed
Password inactive                       : password must be changed
# 此时此帐号的密码创建时间会被改为 1970/1/1 ，所以会有问题！

范例三：尝试以 agetest 登陆的情况
You are required to change your password immediately （root enforced）
WARNING: Your password has expired.
You must change your password now and login again!
Changing password for user agetest.
Changing password for agetest
（current） UNIX password:  &lt;==这个帐号被强制要求必须要改密码！
```

非常有趣吧！你会发现 agetest 这个帐号在第一次登陆时可以使用与帐号同名的密码登陆，但登陆时就会被要求立刻更改密码，更改密码完成后就会被踢出系统。再次登陆时就能够使用新密码登陆了！ 这个功能对学校老师非常有帮助！因为我们不想要知道学生的密码，那么在初次上课时就使用与学号相同的帐号/密码给学生， 让她们登陆时自行设置她们的密码，如此一来就能够避免其他同学随意使用别人的帐号，也能够保证学生知道如何更改自己的密码！

- usermod

所谓这“人有失手，马有乱蹄”，您说是吧？所以啰，当然有的时候会“不小心手滑了一下”在 useradd 的时候加入了错误的设置数据。或者是，在使用 useradd 后，发现某些地方还可以进行细部修改。 此时，当然我们可以直接到 /etc/passwd 或 /etc/shadow 去修改相对应字段的数据， 不过，Linux 也有提供相关的指令让大家来进行帐号相关数据的微调呢～那就是 usermod 啰～

```
[root@study ~]# usermod [-cdegGlsuLU] username
选项与参数：
-c  ：后面接帐号的说明，即 /etc/passwd 第五栏的说明栏，可以加入一些帐号的说明。
-d  ：后面接帐号的主文件夹，即修改 /etc/passwd 的第六栏；
-e  ：后面接日期，格式是 YYYY-MM-DD 也就是在 /etc/shadow 内的第八个字段数据啦！
-f  ：后面接天数，为 shadow 的第七字段。
-g  ：后面接初始群组，修改 /etc/passwd 的第四个字段，亦即是 GID 的字段！
-G  ：后面接次要群组，修改这个使用者能够支持的群组，修改的是 /etc/group 啰～
-a  ：与 -G 合用，可“增加次要群组的支持”而非“设置”喔！
-l  ：后面接帐号名称。亦即是修改帐号名称， /etc/passwd 的第一栏！
-s  ：后面接 Shell 的实际文件，例如 /bin/bash 或 /bin/csh 等等。
-u  ：后面接 UID 数字啦！即 /etc/passwd 第三栏的数据；
-L  ：暂时将使用者的密码冻结，让他无法登陆。其实仅改 /etc/shadow 的密码栏。
-U  ：将 /etc/shadow 密码栏的 ! 拿掉，解冻啦！
```

如果你仔细的比对，会发现 usermod 的选项与 useradd 非常类似！ 这是因为 usermod 也是用来微调 useradd 增加的使用者参数嘛！不过 usermod 还是有新增的选项， 那就是 -L 与 -U ，不过这两个选项其实与 passwd 的 -l, -u 是相同的！而且也不见得会存在所有的 distribution 当中！接下来，让我们谈谈一些变更参数的实例吧！

```
范例一：修改使用者 vbird2 的说明栏，加上“VBird's test”的说明。
[root@study ~]# usermod -c "VBird's test" vbird2
[root@study ~]# grep vbird2 /etc/passwd
vbird2:x:1500:100:VBird's test:/home/vbird2:/bin/bash

范例二：使用者 vbird2 这个帐号在 2015/12/31 失效。
[root@study ~]# usermod -e "2015-12-31" vbird2
[root@study ~]# chage -l vbird2 &#124; grep 'Account expires'
Account expires                        : Dec 31, 2015

范例三：我们创建 vbird3 这个系统帐号时并没有给予主文件夹，请创建他的主文件夹
[root@study ~]# ll -d ~vbird3
ls: cannot access /home/vbird3: No such file or directory  &lt;==确认一下，确实没有主文件夹的
[root@study ~]# cp -a /etc/skel /home/vbird3
[root@study ~]# chown -R vbird3:vbird3 /home/vbird3
[root@study ~]# chmod 700 /home/vbird3
[root@study ~]# ll -a ~vbird3
drwx------. 3 vbird3 vbird3   74 May  4 17:51 .   &lt;==使用者主文件夹权限
drwxr-xr-x. 10 root   root   4096 Jul 20 22:51 ..
-rw-r--r--. 1 vbird3 vbird3   18 Mar  6 06:06 .bash_logout
-rw-r--r--. 1 vbird3 vbird3  193 Mar  6 06:06 .bash_profile
-rw-r--r--. 1 vbird3 vbird3  231 Mar  6 06:06 .bashrc
drwxr-xr-x. 4 vbird3 vbird3   37 May  4 17:51 .mozilla
# 使用 chown -R 是为了连同主文件夹下面的使用者/群组属性都一起变更的意思；
# 使用 chmod 没有 -R ，是因为我们仅要修改目录的权限而非内部文件的权限！
```

- userdel

这个功能就太简单了，目的在删除使用者的相关数据，而使用者的数据有：

- 使用者帐号/密码相关参数：/etc/passwd, /etc/shadow
- 使用者群组相关参数：/etc/group, /etc/gshadow
- 使用者个人文件数据： /home/username, /var/spool/mail/username..

整个指令的语法非常简单：

```
[root@study ~]# userdel [-r] username
选项与参数：
-r  ：连同使用者的主文件夹也一起删除

范例一：删除 vbird2 ，连同主文件夹一起删除
[root@study ~]# userdel -r vbird2
```

这个指令下达的时候要小心了！通常我们要移除一个帐号的时候，你可以手动的将 /etc/passwd 与 /etc/shadow 里头的该帐号取消即可！一般而言，如果该帐号只是“暂时不启用”的话，那么将 /etc/shadow 里头帐号失效日期 （第八字段） 设置为 0 就可以让该帐号无法使用，但是所有跟该帐号相关的数据都会留下来！ 使用 userdel 的时机通常是“你真的确定不要让该用户在主机上面使用任何数据了！”

另外，其实使用者如果在系统上面操作过一阵子了，那么该使用者其实在系统内可能会含有其他文件的。 举例来说，他的邮件信箱 （mailbox） 或者是例行性工作调度 （crontab, 十五章） 之类的文件。 所以，如果想要完整的将某个帐号完整的移除，最好可以在下达 userdel -r username 之前， 先以“ find / -user username ”查出整个系统内属于 username 的文件，然后再加以删除吧！

13.2.2 使用者功能

不论是 useradd/usermod/userdel ，那都是系统管理员所能够使用的指令， 如果我是一般身份使用者，那么我是否除了密码之外，就无法更改其他的数据呢？ 当然不是啦！这里我们介绍几个一般身份使用者常用的帐号数据变更与查询指令啰！

- id

id 这个指令则可以查询某人或自己的相关 UID/GID 等等的信息，他的参数也不少，不过，都不需要记～反正使用 id 就全部都列出啰！ 另外，也回想一下，我们在前一章谈到的循环时，就有用过这个指令喔！ ^_^

```
[root@study ~]# id [username]

范例一：查阅 root 自己的相关 ID 信息！
[root@study ~]# id
uid=0（root） gid=0（root） groups=0（root） context=unconfined_u:unconfined_r:unconfined_t
s0-s0:c0.c1023
# 上面信息其实是同一行的数据！包括会显示 UID/GID 以及支持的所有群组！
# 至于后面那个 context=... 则是 SELinux 的内容，先不要理会他！

范例二：查阅一下 vbird1 吧～
[root@study ~]# id vbird1
uid=1003（vbird1） gid=1004（vbird1） groups=1004（vbird1）

[root@study ~]# id vbird100
id: vbird100: No such user  &lt;== id 这个指令也可以用来判断系统上面有无某帐号！
```

- finger

finger 的中文字面意义是："手指"或者是"指纹"的意思。这个 finger 可以查阅很多使用者相关的信息喔！ 大部分都是在 /etc/passwd 这个文件里面的信息啦！不过，这个指令有点危险，所以新的版本中已经默认不安装这个软件！ 好啦！现在继续来安装软件先～记得第九章 dos2unix 的安装方式！ 假设你已经将光驱或光盘镜像文件挂载在 /mnt 下面了，所以：

```
[root@study ~]# df -hT /mnt
Filesystem     Type     Size  Used Avail Use% Mounted on
/dev/sr0       iso9660  7.1G  7.1G     0 100% /mnt    # 先确定是有挂载光盘的啦！

[root@study ~]# rpm -ivh /mnt/Packages/finger-[0-9]*
```

我们就先来检查检查使用者信息吧！

```
[root@study ~]# finger [-s] username
选项与参数：
-s  ：仅列出使用者的帐号、全名、终端机代号与登陆时间等等；
-m  ：列出与后面接的帐号相同者，而不是利用部分比对 （包括全名部分）

范例一：观察 vbird1 的使用者相关帐号属性
[root@study ~]# finger vbird1
Login: vbird1                           Name:
Directory: /home/vbird1                 Shell: /bin/bash
Never logged in.
No mail.
No Plan.
```

由于 finger 类似指纹的功能，他会将使用者的相关属性列出来！如上表所示，其实他列出来的几乎都是 /etc/passwd 文件里面的东西。列出的信息说明如下：

- Login：为使用者帐号，亦即 /etc/passwd 内的第一字段；
- Name：为全名，亦即 /etc/passwd 内的第五字段（或称为注解）；
- Directory：就是主文件夹了；
- Shell：就是使用的 Shell 文件所在；
- Never logged in.：figner 还会调查使用者登陆主机的情况喔！
- No mail.：调查 /var/spool/mail 当中的信箱数据；
- No Plan.：调查 ~vbird1/.plan 文件，并将该文件取出来说明！

不过是否能够查阅到 Mail 与 Plan 则与权限有关了！因为 Mail / Plan 都是与使用者自己的权限设置有关， root 当然可以查阅到使用者的这些信息，但是 vbird1 就不见得能够查到 vbird3 的信息， 因为 /var/spool/mail/vbird3 与 /home/vbird3/ 的权限分别是 660, 700 ，那 vbird1 当然就无法查阅的到！ 这样解释可以理解吧？此外，我们可以创建自己想要执行的预定计划，当然，最多是给自己看的！可以这样做：

```
范例二：利用 vbird1 创建自己的计划档
[vbird1@study ~]$ echo "I will study Linux during this year." &gt; ~/.plan
[vbird1@study ~]$ finger vbird1
Login: vbird1                           Name:
Directory: /home/vbird1                 Shell: /bin/bash
Last login Mon Jul 20 23:06 （CST） on pts/0
No mail.
Plan:
I will study Linux during this year.

范例三：找出目前在系统上面登陆的使用者与登陆时间
[vbird1@study ~]$ finger
Login     Name       Tty      Idle  Login Time   Office     Office Phone   Host
dmtsai    dmtsai     tty2      11d  Jul  7 23:07
dmtsai    dmtsai     pts/0          Jul 20 17:59
```

在范例三当中，我们发现输出的信息还会有 Office, Office Phone 等信息，那这些信息要如何记录呢？ 下面我们会介绍 chfn 这个指令！来看看如何修改使用者的 finger 数据吧！

- chfn

chfn 有点像是： change finger 的意思！这玩意的使用方法如下：

```
[root@study ~]# chfn [-foph] [帐号名]
选项与参数：
-f  ：后面接完整的大名；
-o  ：您办公室的房间号码；
-p  ：办公室的电话号码；
-h  ：家里的电话号码！

范例一：vbird1 自己更改一下自己的相关信息！
[vbird1@study ~]$ chfn
Changing finger information for vbird1.
Name []: VBird Tsai test         &lt;==输入你想要呈现的全名
Office []: DIC in KSU            &lt;==办公室号码
Office Phone []: 06-2727175#356  &lt;==办公室电话
Home Phone []: 06-1234567        &lt;==家里电话号码

Password:  &lt;==确认身份，所以输入自己的密码
Finger information changed.

[vbird1@study ~]$ grep vbird1 /etc/passwd
vbird1:x:1003:1004:VBird Tsai test,DIC in KSU,06-2727175#356,06-1234567:/home/vbird1:/bin
# 其实就是改到第五个字段，该字段里面用多个“ , ”分隔就是了！

[vbird1@study ~]$ finger vbird1
Login: vbird1                           Name: VBird Tsai test
Directory: /home/vbird1                 Shell: /bin/bash
Office: DIC in KSU, 06-2727175#356      Home Phone: 06-1234567
Last login Mon Jul 20 23:12 （CST） on pts/0
No mail.
Plan:
I will study Linux during this year.
# 就是上面特殊字体呈现的那些地方是由 chfn 所修改出来的！
```

这个指令说实在的，除非是你的主机有很多的用户，否则倒真是用不着这个程序！这就有点像是 bbs 里头更改你“个人属性”的那一个数据啦！不过还是可以自己玩一玩！尤其是用来提醒自己相关数据啦！ ^_^

- chsh

这就是 change shell 的简写！使用方法就更简单了！

```
[vbird1@study ~]$ chsh [-ls]
选项与参数：
-l  ：列出目前系统上面可用的 shell ，其实就是 /etc/shells 的内容！
-s  ：设置修改自己的 Shell 啰

范例一：用 vbird1 的身份列出系统上所有合法的 shell，并且指定 csh 为自己的 shell
[vbird1@study ~]$ chsh -l
/bin/sh
/bin/bash
/sbin/nologin   &lt;==所谓：合法不可登陆的 Shell 就是这玩意！
/usr/bin/sh
/usr/bin/bash
/usr/sbin/nologin
/bin/tcsh
/bin/csh        &lt;==这就是 C shell 啦！
# 其实上面的信息就是我们在 [bash](../Text/index.html) 中谈到的 /etc/shells 啦！

[vbird1@study ~]$ chsh -s /bin/csh; grep vbird1 /etc/passwd
Changing shell for vbird1.
Password:  &lt;==确认身份，请输入 vbird1 的密码
Shell changed.
vbird1:x:1003:1004:VBird Tsai test,DIC in KSU,06-2727175#356,06-1234567:/home/vbird1:/bin

[vbird1@study ~]$ chsh -s /bin/bash
# 测试完毕后，立刻改回来！

[vbird1@study ~]$ ll $（which chsh）
-rws--x--x. 1 root root 23856 Mar  6 13:59 /bin/chsh
```

不论是 chfn 与 chsh ，都是能够让一般使用者修改 /etc/passwd 这个系统文件的！所以你猜猜，这两个文件的权限是什么？ 一定是 SUID 的功能啦！看到这里，想到前面！ 这就是 Linux 的学习方法～ ^_^

13.2.3 新增与移除群组

OK！了解了帐号的新增、删除、更动与查询后，再来我们可以聊一聊群组的相关内容了。 基本上，群组的内容都与这两个文件有关：/etc/group, /etc/gshadow。 群组的内容其实很简单，都是上面两个文件的新增、修改与移除而已， 不过，如果再加上有效群组的概念，那么 newgrp 与 gpasswd 则不可不知呢！

- groupadd

```
[root@study ~]# groupadd [-g gid] [-r] 群组名称
选项与参数：
-g  ：后面接某个特定的 GID ，用来直接给予某个 GID ～
-r  ：创建系统群组啦！与 /etc/login.defs 内的 GID_MIN 有关。

范例一：新建一个群组，名称为 group1
[root@study ~]# groupadd group1
[root@study ~]# grep group1 /etc/group /etc/gshadow
/etc/group:group1:x:1503:
/etc/gshadow:group1:!::
# 群组的 GID 也是会由 1000 以上最大 GID+1 来决定！
```

曾经有某些版本的教育训练手册谈到，为了让使用者的 UID/GID 成对，她们建议新建的与使用者私有群组无关的其他群组时，使用小于 1000 以下的 GID 为宜。 也就是说，如果要创建群组的话，最好能够使用“ groupadd -r 群组名”的方式来创建啦！ 不过，这见仁见智啦！看你自己的抉择啰！

- groupmod

跟 usermod 类似的，这个指令仅是在进行 group 相关参数的修改而已。

```
[root@study ~]# groupmod [-g gid] [-n group_name] 群组名
选项与参数：
-g  ：修改既有的 GID 数字；
-n  ：修改既有的群组名称

范例一：将刚刚上个指令创建的 group1 名称改为 mygroup ， GID 为 201
[root@study ~]# groupmod -g 201 -n mygroup group1
[root@study ~]# grep mygroup /etc/group /etc/gshadow
/etc/group:mygroup:x:201:
/etc/gshadow:mygroup:!::
```

不过，还是那句老话，不要随意的更动 GID ，容易造成系统资源的错乱喔！

- groupdel

呼呼！ groupdel 自然就是在删除群组的啰～用法很简单：

```
[root@study ~]# groupdel [groupname]

范例一：将刚刚的 mygroup 删除！
[root@study ~]# groupdel mygroup

范例二：若要删除 vbird1 这个群组的话？
[root@study ~]# groupdel vbird1
groupdel: cannot remove the primary group of user 'vbird1'
```

为什么 mygroup 可以删除，但是 vbird1 就不能删除呢？原因很简单，“有某个帐号（/etc/passwd） 的 initial group 使用该群组！” 如果查阅一下，你会发现在 /etc/passwd 内的 vbird1 第四栏的 GID 就是 /etc/group 内的 vbird1 那个群组的 GID ，所以啰，当然无法删除～否则 vbird1 这个使用者登陆系统后， 就会找不到 GID ，那可是会造成很大的困扰的！那么如果硬要删除 vbird1 这个群组呢？ 你“必须要确认 /etc/passwd 内的帐号没有任何人使用该群组作为 initial group ”才行喔！所以，你可以：

- 修改 vbird1 的 GID ，或者是：
- 删除 vbird1 这个使用者。

- gpasswd：群组管理员功能

如果系统管理员太忙碌了，导致某些帐号想要加入某个专案时找不到人帮忙！这个时候可以创建“群组管理员”喔！ 什么是群组管理员呢？就是让某个群组具有一个管理员，这个群组管理员可以管理哪些帐号可以加入/移出该群组！ 那要如何“创建一个群组管理员”呢？就得要通过 gpasswd 啰！

```
# 关于系统管理员（root）做的动作：
[root@study ~]# gpasswd groupname
[root@study ~]# gpasswd [-A user1,...] [-M user3,...] groupname
[root@study ~]# gpasswd [-rR] groupname
选项与参数：
    ：若没有任何参数时，表示给予 groupname 一个密码（/etc/gshadow）
-A  ：将 groupname 的主控权交由后面的使用者管理（该群组的管理员）
-M  ：将某些帐号加入这个群组当中！
-r  ：将 groupname 的密码移除
-R  ：让 groupname 的密码栏失效

# 关于群组管理员（Group administrator）做的动作：
[someone@study ~]$ gpasswd [-ad] user groupname
选项与参数：
-a  ：将某位使用者加入到 groupname 这个群组当中！
-d  ：将某位使用者移除出 groupname 这个群组当中。

范例一：创建一个新群组，名称为 testgroup 且群组交由 vbird1 管理：
[root@study ~]# groupadd testgroup  &lt;==先创建群组
[root@study ~]# gpasswd testgroup   &lt;==给这个群组一个密码吧！
Changing the password for group testgroup
New Password:
Re-enter new password:
# 输入两次密码就对了！
[root@study ~]# gpasswd -A vbird1 testgroup  &lt;==加入群组管理员为 vbird1
[root@study ~]# grep testgroup /etc/group /etc/gshadow
/etc/group:testgroup:x:1503:
/etc/gshadow:testgroup:$6$MnmChP3D$mrUn.Vo.buDjObMm8F2emTkvGSeuWikhRzaKHxpJ...:vbird1:
# 很有趣吧！此时 vbird1 则拥有 testgroup 的主控权喔！身份有点像板主啦！

范例二：以 vbird1 登陆系统，并且让他加入 vbird1, vbird3 成为 testgroup 成员
[vbird1@study ~]$ id
uid=1003（vbird1） gid=1004（vbird1） groups=1004（vbird1） ...
# 看得出来，vbird1 尚未加入 testgroup 群组喔！

[vbird1@study ~]$ gpasswd -a vbird1 testgroup
[vbird1@study ~]$ gpasswd -a vbird3 testgroup
[vbird1@study ~]$ grep testgroup /etc/group
testgroup:x:1503:vbird1,vbird3
```

很有趣的一个小实验吧！我们可以让 testgroup 成为一个可以公开的群组，然后创建起群组管理员， 群组管理员可以有多个。在这个案例中，我将 vbird1 设置为 testgroup 的群组管理员，所以 vbird1 就可以自行增加群组成员啰～呼呼！然后，该群组成员就能够使用 newgrp 啰～

13.2.4 帐号管理实例

帐号管理不是随意创建几个帐号就算了！有时候我们需要考虑到一部主机上面可能有多个帐号在协同工作！ 举例来说，在大学任教时，我们学校的专题生是需要分组的，这些同一组的同学间必须要能够互相修改对方的数据文件， 但是同时这些同学又需要保留自己的私密数据，因此直接公开主文件夹是不适宜的。那该如何是好？ 为此，我们下面提供几个例子来让大家思考看看啰：

任务一：单纯的完成上头交代的任务，假设我们需要的帐号数据如下，你该如何实作？

帐号名称	帐号全名	支持次要群组	是否可登陆主机	密码
myuser1	1st user	mygroup1	可以	password
myuser2	2nd user	mygroup1	可以	password
myuser3	3rd user	无额外支持	不可以	password

处理的方法如下所示：

```
# 先处理帐号相关属性的数据：
[root@study ~]# groupadd mygroup1
[root@study ~]# useradd -G mygroup1 -c "1st user" myuser1
[root@study ~]# useradd -G mygroup1 -c "2nd user" myuser2
[root@study ~]# useradd -c "3rd user" -s /sbin/nologin myuser3

# 再处理帐号的密码相关属性的数据：
[root@study ~]# echo "password" &#124; passwd --stdin myuser1
[root@study ~]# echo "password" &#124; passwd --stdin myuser2
[root@study ~]# echo "password" &#124; passwd --stdin myuser3
```

要注意的地方主要有：myuser1 与 myuser2 都有支持次要群组，但该群组不见得会存在，因此需要先手动创建他！ 然后 myuser3 是“不可登陆系统”的帐号，因此需要使用 /sbin/nologin 这个 shell 来给予，这样该帐号就无法登陆啰！ 这样是否理解啊！接下来再来讨论比较难一些的环境！如果是专题环境该如何制作？

任务二：我的使用者 pro1, pro2, pro3 是同一个专案计划的开发人员，我想要让这三个用户在同一个目录下面工作， 但这三个用户还是拥有自己的主文件夹与基本的私有群组。假设我要让这个专案计划在 /srv/projecta 目录下开发， 可以如何进行？

```
# 1\. 假设这三个帐号都尚未创建，可先创建一个名为 projecta 的群组，
#    再让这三个用户加入其次要群组的支持即可：
[root@study ~]# groupadd projecta
[root@study ~]# useradd -G projecta -c "projecta user" pro1
[root@study ~]# useradd -G projecta -c "projecta user" pro2
[root@study ~]# useradd -G projecta -c "projecta user" pro3
[root@study ~]# echo "password" &#124; passwd --stdin pro1
[root@study ~]# echo "password" &#124; passwd --stdin pro2
[root@study ~]# echo "password" &#124; passwd --stdin pro3

# 2\. 开始创建此专案的开发目录：
[root@study ~]# mkdir /srv/projecta
[root@study ~]# chgrp projecta /srv/projecta
[root@study ~]# chmod 2770 /srv/projecta
[root@study ~]# ll -d /srv/projecta
drwxrws---. 2 root projecta 6 Jul 20 23:32 /srv/projecta
```

由于此专案计划只能够给 pro1, pro2, pro3 三个人使用，所以 /srv/projecta 的权限设置一定要正确才行！ 所以该目录群组一定是 projecta ，但是权限怎么会是 2770 呢还记得第六章谈到的 SGID 吧？为了让三个使用者能够互相修改对方的文件， 这个 SGID 是必须要存在的喔！ 如果连这里都能够理解，嘿嘿！您的帐号管理已经有一定程度的概念啰！ ^_^

但接下来有个困扰的问题发生了！假如任务一的 myuser1 是 projecta 这个专案的助理，他需要这个专案的内容， 但是他“不可以修改”专案目录内的任何数据！那该如何是好？你或许可以这样做：

- 将 myuser1 加入 projecta 这个群组的支持，但是这样会让 myuser1 具有完整的 /srv/projecta 的使用权限， myuser1 是可以删除该目录下的任何数据的！这样是有问题的；
- 将 /srv/projecta 的权限改为 2775 ，让 myuser1 可以进入查阅数据。但此时会发生所有其他人均可进入该目录查阅的困扰！ 这也不是我们要的环境。

眞要命！传统的 Linux 权限无法针对某个个人设置专属的权限吗？其实是可以啦！接下来我们就来谈谈这个功能吧！

13.2.5 使用外部身份认证系统

在谈 ACL 之前，我们再来谈一个概念性的操作～因为我们目前没有服务器可供练习....

有时候，除了本机的帐号之外，我们可能还会使用到其他外部的身份验证服务器所提供的验证身份的功能！举例来说， windows 下面有个很有名的身份验证系统，称为 Active Directory（AD）的东西，还有 Linux 为了提供不同主机使用同一组帐号密码， 也会使用到 LDAP, NIS 等服务器提供的身份验证等等！

如果你的 Linux 主机要使用到上面提到的这些外部身份验证系统时，可能就得要额外的设置一些数据了！ 为了简化使用者的操作流程，所以 CentOS 提供一只名为 authconfig-tui 的指令给我们参考，这个指令的执行结果如下：

图13.2.1、使用外部身份验证服务器的方式

你可以在该画面中使用 [tab] 按钮在各个项目中间切换，不过，因为我们没有适用的服务器可以测试，因此这里仅是提供一个参考的依据， 未来如果谈到服务器章节时，你要如果谈到服务器章节时，服器有印象，处理外部身份验证的方式可以通过 authconfig-tui 就好了！ 上图中最多可供操作的，大概仅有支持 MD5 这个早期的密码格式就是了！此外，不要随便将已经启用的项目 （上头有星号 * 的项目） 取消喔！ 可能某些帐号会失效...

13.3 主机的细部权限规划：ACL 的使用

从第五章开始，我们就一直强调 Linux 的权限概念是非常重要的！ 但是传统的权限仅有三种身份 （owner, group, others） 搭配三种权限 （r,w,x） 而已，并没有办法单纯的针对某一个使用者或某一个群组来设置特定的权限需求，例如前一小节最后的那个任务！ 此时就得要使用 ACL 这个机制啦！这玩意挺有趣的，下面我们就来谈一谈：

13.3.1 什么是 ACL 与如何支持启动 ACL

ACL 是 Access Control List 的缩写，主要的目的是在提供传统的 owner,group,others 的 read,write,execute 权限之外的细部权限设置。ACL 可以针对单一使用者，单一文件或目录来进行 r,w,x 的权限规范，对于需要特殊权限的使用状况非常有帮助。

那 ACL 主要可以针对哪些方面来控制权限呢？他主要可以针对几个项目：

- 使用者 （user）：可以针对使用者来设置权限；
- 群组 （group）：针对群组为对象来设置其权限；
- 默认属性 （mask）：还可以针对在该目录下在创建新文件/目录时，规范新数据的默认权限；

也就是说，如果你有一个目录，需要给一堆人使用，每个人或每个群组所需要的权限并不相同时，在过去，传统的 Linux 三种身份的三种权限是无法达到的， 因为基本上，传统的 Linux 权限只能针对一个用户、一个群组及非此群组的其他人设置权限而已，无法针对单一用户或个人来设计权限。 而 ACL 就是为了要改变这个问题啊！好了，稍微了解之后，再来看看如何让你的文件系统可以支持 ACL 吧！

- 如何启动 ACL

事实上，原本 ACL 是 unix-like 操作系统的额外支持项目，但因为近年以来 Linux 系统对权限细部设置的热切需求， 因此目前 ACL 几乎已经默认加入在所有常见的 Linux 文件系统的挂载参数中 （ext2/ext3/ext4/xfs等等）！所以你无须进行任何动作， ACL 就可以被你使用啰！不过，如果你不放心系统是否真的有支持 ACL 的话，那么就来检查一下核心挂载时显示的信息吧！

```
[root@study ~]# dmesg &#124; grep -i acl
[    0.330377] systemd[1]: systemd 208 running in system mode. （+PAM +LIBWRAP +AUDIT
+SELINUX +IMA +SYSVINIT +LIBCRYPTSETUP +GCRYPT +ACL +XZ）
[    0.878265] SGI XFS with ACLs, security attributes, large block/inode numbers, no
debug enabled
```

瞧！至少 xfs 已经支持这个 ACL 的功能啰！

13.3.2 ACL 的设置技巧： getfacl, setfacl

好了，既然知道我们的 filesystem 有支持 ACL 之后，接下来该如何设置与观察 ACL 呢？ 很简单，利用这两个指令就可以了：

- getfacl：取得某个文件/目录的 ACL 设置项目；
- setfacl：设置某个目录/文件的 ACL 规范。

先让我们来瞧一瞧 setfacl 如何使用吧！

- setfacl 指令用法介绍及最简单的“ u:帐号:权限 ”设置

```
[root@study ~]# setfacl [-bkRd] [{-m&#124;-x} acl参数] 目标文件名
选项与参数：
-m ：设置后续的 acl 参数给文件使用，不可与 -x 合用；
-x ：删除后续的 acl 参数，不可与 -m 合用；
-b ：移除“所有的” ACL 设置参数；
-k ：移除“默认的” ACL 参数，关于所谓的“默认”参数于后续范例中介绍；
-R ：递回设置 acl ，亦即包括次目录都会被设置起来；
-d ：设置“默认 acl 参数”的意思！只对目录有效，在该目录新建的数据会引用此默认值
```

上面谈到的是 acl 的选项功能，那么如何设置 ACL 的特殊权限呢？特殊权限的设置方法有很多， 我们先来谈谈最常见的，就是针对单一使用者的设置方式：

```
# 1\. 针对特定使用者的方式：
# 设置规范：“ u:[使用者帐号列表]:[rwx] ”，例如针对 vbird1 的权限规范 rx ：
[root@study ~]# touch acl_test1
[root@study ~]# ll acl_test1
-rw-r--r--. 1 root root 0 Jul 21 17:33 acl_test1
[root@study ~]# setfacl -m u:vbird1:rx acl_test1
[root@study ~]# ll acl_test1
-rw-r-xr--+ 1 root root 0 Jul 21 17:33 acl_test1
# 权限部分多了个 + ，且与原本的权限 （644） 看起来差异很大！但要如何查阅呢？

[root@study ~]# setfacl -m u::rwx acl_test1
[root@study ~]# ll acl_test1
-rwxr-xr--+ 1 root root 0 Jul 21 17:33 acl_test1
# 设置值中的 u 后面无使用者列表，代表设置该文件拥有者，所以上面显示 root 的权限成为 rwx 了！
```

上述动作为最简单的 ACL 设置，利用“ u:使用者:权限 ”的方式来设置的啦！设置前请加上 -m 这个选项。 如果一个文件设置了 ACL 参数后，他的权限部分就会多出一个 + 号了！但是此时你看到的权限与实际权限可能就会有点误差！ 那要如何观察呢？就通过 getfacl 吧！

- getfacl 指令用法

```
[root@study ~]# getfacl filename
选项与参数：
getfacl 的选项几乎与 setfacl 相同！所以鸟哥这里就免去了选项的说明啊！

# 请列出刚刚我们设置的 acl_test1 的权限内容：
[root@study ~]# getfacl acl_test1
# file: acl_test1   &lt;==说明文档名而已！
# owner: root       &lt;==说明此文件的拥有者，亦即 ls -l 看到的第三使用者字段
# group: root       &lt;==此文件的所属群组，亦即 ls -l 看到的第四群组字段
user::rwx           &lt;==使用者列表栏是空的，代表文件拥有者的权限
user:vbird1:r-x     &lt;==针对 vbird1 的权限设置为 rx ，与拥有者并不同！
group::r--          &lt;==针对文件群组的权限设置仅有 r
mask::r-x           &lt;==此文件默认的有效权限 （mask）
other::r--          &lt;==其他人拥有的权限啰！
```

上面的数据非常容易查阅吧？显示的数据前面加上 # 的，代表这个文件的默认属性，包括文件名、文件拥有者与文件所属群组。 下面出现的 user, group, mask, other 则是属于不同使用者、群组与有效权限（mask）的设置值。 以上面的结果来看，我们刚刚设置的 vbird1 对于这个文件具有 r 与 x 的权限啦！这样看的懂吗？ 如果看的懂的话，接下来让我们在测试其他类型的 setfacl 设置吧！

- 特定的单一群组的权限设置：“ g:群组名:权限 ”

```
# 2\. 针对特定群组的方式：
# 设置规范：“ g:[群组列表]:[rwx] ”，例如针对 mygroup1 的权限规范 rx ：
[root@study ~]# setfacl -m g:mygroup1:rx acl_test1
[root@study ~]# getfacl acl_test1
# file: acl_test1
# owner: root
# group: root
user::rwx
user:vbird1:r-x
group::r--
group:mygroup1:r-x  &lt;==这里就是新增的部分！多了这个群组的权限设置！
mask::r-x
other::r--
```

- 针对有效权限设置：“ m:权限 ”

基本上，群组与使用者的设置并没有什么太大的差异啦！如上表所示，非常容易了解意义。 不过，你应该会觉得奇怪的是， 那个 mask 是什么东西啊？其实他有点像是“有效权限”的意思！他的意义是： 使用者或群组所设置的权限必须要存在于 mask 的权限设置范围内才会生效，此即“有效权限 （effective permission）” 我们举个例子来看，如下所示：

```
# 3\. 针对有效权限 mask 的设置方式：
# 设置规范：“ m:[rwx] ”，例如针对刚刚的文件规范为仅有 r ：
[root@study ~]# setfacl -m m:r acl_test1
[root@study ~]# getfacl acl_test1
# file: acl_test1
# owner: root
# group: root
user::rwx
user:vbird1:r-x        #effective:r-- &lt;==vbird1+mask均存在者，仅有 r 而已，x 不会生效
group::r--
group:mygroup1:r-x     #effective:r--
mask::r--
other::r--
```

您瞧，vbird1 与 mask 的集合发现仅有 r 存在，因此 vbird1 仅具有 r 的权限而已，并不存在 x 权限！这就是 mask 的功能了！我们可以通过使用 mask 来规范最大允许的权限，就能够避免不小心开放某些权限给其他使用者或群组了。 不过，通常鸟哥都是将 mask 设置为 rwx 啦！ 然后再分别依据不同的使用者/群组去规范她们的权限就是了。

例题：将前一小节任务二中 /srv/projecta 这个目录，让 myuser1 可以进入查阅，但 myuser1 不具有修改的权力。答：由于 myuser1 是独立的使用者与群组，因此无法使用传统的 Linux 权限设置。此时使用 ACL 的设置如下：

```
# 1\. 先测试看看，使用 myuser1 能否进入该目录？
[myuser1@study ~]$ cd /srv/projecta
-bash: cd: /srv/projecta: Permission denied  &lt;==确实不可进入！

# 2\. 开始用 root 的身份来设置一下该目录的权限吧！
[root@study ~]# setfacl -m u:myuser1:rx /srv/projecta
[root@study ~]# getfacl /srv/projecta
# file: srv/projecta
# owner: root
# group: projecta
# flags: -s-
user::rwx
user:myuser1:r-x  &lt;==还是要看看有没有设置成功喔！
group::rwx
mask::rwx
other::---

# 3\. 还是得要使用 myuser1 去测试看看结果！
[myuser1@study ~]$ cd /srv/projecta
[myuser1@study projecta]$ ll -a
drwxrws---+ 2 root projecta 4096 Feb 27 11:29 .  &lt;==确实可以查询文件名
drwxr-xr-x  4 root root     4096 Feb 27 11:29 ..

[myuser1@study projecta]$ touch testing
touch: cannot touch `testing': Permission denied &lt;==确实不可以写入！
```

请注意，上述的 1, 3 步骤使用 myuser1 的身份，2步骤才是使用 root 去设置的！

上面的设置我们就完成了之前任务二的后续需求喔！这么简单呢！接下来让我们来测试一下，如果我用 root 或者是 pro1 的身份去 /srv/projecta 增加文件或目录时，该文件或目录是否能够具有 ACL 的设置？ 意思就是说，ACL 的权限设置是否能够被次目录所“继承？”先试看看：

```
[root@study ~]# cd /srv/projecta
[root@study ~]# touch abc1
[root@study ~]# mkdir abc2
[root@study ~]# ll -d abc*
-rw-r--r--. 1 root projecta 0 Jul 21 17:49 abc1
drwxr-sr-x. 2 root projecta 6 Jul 21 17:49 abc2
```

你可以明显的发现，权限后面都没有 + ，代表这个 acl 属性并没有继承喔！如果你想要让 acl 在目录下面的数据都有继承的功能，那就得如下这样做了！

- 使用默认权限设置目录未来文件的 ACL 权限继承“ d:[u|g]:[user|group]:权限 ”

```
# 4\. 针对默认权限的设置方式：
# 设置规范：“ d:[ug]:使用者列表:[rwx] ”

# 让 myuser1 在 /srv/projecta 下面一直具有 rx 的默认权限！
[root@study ~]# setfacl -m d:u:myuser1:rx /srv/projecta
[root@study ~]# getfacl /srv/projecta
# file: srv/projecta
# owner: root
# group: projecta
# flags: -s-
user::rwx
user:myuser1:r-x
group::rwx
mask::rwx
other::---
default:user::rwx
default:user:myuser1:r-x
default:group::rwx
default:mask::rwx
default:other::---

[root@study ~]# cd /srv/projecta
[root@study projecta]# touch zzz1
[root@study projecta]# mkdir zzz2
[root@study projecta]# ll -d zzz*
-rw-rw----+ 1 root projecta 0 Jul 21 17:50 zzz1
drwxrws---+ 2 root projecta 6 Jul 21 17:51 zzz2
# 看吧！确实有继承喔！然后我们使用 getfacl 再次确认看看！

[root@study projecta]# getfacl zzz2
# file: zzz2
# owner: root
# group: projecta
# flags: -s-
user::rwx
user:myuser1:r-x
group::rwx
mask::rwx
other::---
default:user::rwx
default:user:myuser1:r-x
default:group::rwx
default:mask::rwx
default:other::---
```

通过这个“针对目录来设置的默认 ACL 权限设置值”的项目，我们可以让这些属性继承到次目录下面呢！ 非常方便啊！那如果想要让 ACL 的属性全部消失又要如何处理？通过“ setfacl -b 文件名 ”即可啦！ 太简单了！鸟哥就不另外介绍了！请自行测试测试吧！

问：针对刚刚的 /srv/projecta 目录的权限设置中，我需要 1）取消 myuser1 的设置（连同默认值），以及 2）我不能让 pro3 这个用户使用该目录，亦即 pro3 在该目录下无任何权限，该如何设置？答：取消全部的 ACL 设置可以使用 -b 来处理，但单一设置值的取消，就得要通过 -x 才行了！所以你应该这样作：

```
# 1.1 找到针对 myuser1 的设置值
[root@study ~]# getfacl /srv/projecta &#124; grep myuser1
user:myuser1:r-x
default:user:myuser1:r-x

# 1.2 针对每个设置值来处理，注意，取消某个帐号的 ACL 时，不需要加上权限项目！
[root@study ~]# setfacl -x u:myuser1 /srv/projecta
[root@study ~]# setfacl -x d:u:myuser1 /srv/projecta

# 2.1 开始让 pro3 这个用户无法使用该目录啰！
[root@study ~]# setfacl -m u:pro3:- /srv/projecta
```

只需要留意，当设置一个用户/群组没有任何权限的 ACL 语法中，在权限的字段不可留白，而是应该加上一个减号 （-） 才是正确的作法！

13.4 使用者身份切换

什么？在 Linux 系统当中还要作身份的变换？这是为啥？可能有下面几个原因啦！

- 使用一般帐号：系统平日操作的好习惯

事实上，为了安全的缘故，一些老人家都会建议你，尽量以一般身份使用者来操作 Linux 的日常作业！等到需要设置系统环境时， 才变换身份成为 root 来进行系统管理，相对比较安全啦！避免作错一些严重的指令，例如恐怖的“ rm -rf / ”（千万作不得！）

- 用较低权限启动系统服务

相对于系统安全，有的时候，我们必须要以某些系统帐号来进行程序的执行。 举例来说，Linux 主机上面的一套软件，名称为 apache ，我们可以额外创建一个名为 apache 的使用者来启动 apache 软件啊，如此一来，如果这个程序被攻破，至少系统还不至于就损毁了～

- 软件本身的限制

在远古时代的 telnet 程序中，该程序默认是不许使用 root 的身份登陆的，telnet 会判断登陆者的 UID， 若 UID 为 0 的话，那就直接拒绝登陆了。所以，你只能使用一般使用者来登陆 Linux 服务器。 此外， ssh [3] 也可以设置拒绝 root 登陆喔！那如果你有系统设置需求该如何是好啊？就变换身份啊！

由于上述考虑，所以我们都是使用一般帐号登陆系统的，等有需要进行系统维护或软件更新时才转为 root 的身份来动作。 那如何让一般使用者转变身份成为 root 呢？主要有两种方式喔：

- 以“ su - ”直接将身份变成 root 即可，但是这个指令却需要 root 的密码，也就是说，如果你要以 su 变成 root 的话，你的一般使用者就必须要有 root 的密码才行；
- 以“ sudo 指令 ”执行 root 的指令串，由于 sudo 需要事先设置妥当，且 sudo 需要输入使用者自己的密码， 因此多人共管同一部主机时， sudo 要比 su 来的好喔！至少 root 密码不会流出去！

下面我们就来说一说 su 跟 sudo 的用法啦！

13.4.1 su

su 是最简单的身份切换指令了，他可以进行任何身份的切换唷！方法如下：

```
[root@study ~]# su [-lm] [-c 指令] [username]
选项与参数：
-    ：单纯使用 - 如“ su - ”代表使用 login-shell 的变量文件读取方式来登陆系统；
       若使用者名称没有加上去，则代表切换为 root 的身份。
-l   ：与 - 类似，但后面需要加欲切换的使用者帐号！也是 login-shell 的方式。
-m   ：-m 与 -p 是一样的，表示“使用目前的环境设置，而不读取新使用者的配置文件”
-c   ：仅进行一次指令，所以 -c 后面可以加上指令喔！
```

上表的解释当中有出现之前第十章谈过的 login-shell 配置文件读取方式，如果你忘记那是啥东西， 请先回去第十章瞧瞧再回来吧！这个 su 的用法当中，有没有加上那个减号“ - ”差很多喔！ 因为涉及 login-shell 与 non-login shell 的变量读取方法。这里让我们以一个小例子来说明吧！

```
范例一：假设你原本是 dmtsai 的身份，想要使用 non-login shell 的方式变成 root
[dmtsai@study ~]$ su       &lt;==注意提示字符，是 dmtsai 的身份喔！
Password:                  &lt;==这里输入 root 的密码喔！
[root@study dmtsai]# id    &lt;==提示字符的目录是 dmtsai 喔！
uid=0（root） gid=0（root） groups=0（root） context=unconf....  &lt;==确实是 root 的身份！
[root@study dmtsai]# env &#124; grep 'dmtsai'
USER=dmtsai                                       &lt;==竟然还是 dmtsai 这家伙！
PATH=...:/home/dmtsai/.local/bin:/home/dmtsai/bin  &lt;==这个影响最大！
MAIL=/var/spool/mail/dmtsai                       &lt;==收到的 mailbox 是 vbird1
PWD=/home/dmtsai                                  &lt;==并非 root 的主文件夹
LOGNAME=dmtsai
# 虽然你的 UID 已经是具有 root 的身份，但是看到上面的输出讯息吗？
# 还是有一堆变量为原本 dmtsai 的身份，所以很多数据还是无法直接利用。
[root@study dmtsai]# exit   &lt;==这样可以离开 su 的环境！
```

单纯使用“ su ”切换成为 root 的身份，读取的变量设置方式为 non-login shell 的方式，这种方式很多原本的变量不会被改变， 尤其是我们之前谈过很多次的 PATH 这个变量，由于没有改变成为 root 的环境， 因此很多 root 惯用的指令就只能使用绝对路径来执行咯。其他的还有 MAIL 这个变量，你输入 mail 时， 收到的邮件竟然还是 dmtsai 的，而不是 root 本身的邮件！是否觉得很奇怪啊！所以切换身份时，请务必使用如下的范例二：

```
范例二：使用 login shell 的方式切换为 root 的身份并观察变量
[dmtsai@study ~]$ su -
Password:   &lt;==这里输入 root 的密码喔！
[root@study ~]# env &#124; grep root
USER=root
MAIL=/var/spool/mail/root
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
PWD=/root
HOME=/root
LOGNAME=root
# 了解差异了吧？下次变换成为 root 时，记得最好使用 su - 喔！
[root@study ~]# exit   &lt;==这样可以离开 su 的环境！
```

上述的作法是让使用者的身份变成 root 并开始操作系统，如果想要离开 root 的身份则得要利用 exit 离开才行。 那我如果只是想要执行“一个只有 root 才能进行的指令，且执行完毕就恢复原本的身份”呢？那就可以加上 -c 这个选项啰！ 请参考下面范例三！

```
范例三：dmtsai 想要执行“ head -n 3 /etc/shadow ”一次，且已知 root 密码
[dmtsai@study ~]$ head -n 3 /etc/shadow
head: cannot open `/etc/shadow' for reading: Permission denied
[dmtsai@study ~]$ su - -c "head -n 3 /etc/shadow"
Password: &lt;==这里输入 root 的密码喔！
root:$6$wtbCCce/PxMeE5wm$KE2IfSJr.YLP7Rcai6oa/T7KFhOYO62vDnqfLw85...:16559:0:99999:7:::
bin:*:16372:0:99999:7:::
daemon:*:16372:0:99999:7:::
[dmtsai@study ~]$ &lt;==注意看，身份还是 dmtsai 喔！继续使用旧的身份进行系统操作！
```

由于 /etc/shadow 权限的关系，该文件仅有 root 可以查阅。为了查阅该文件，所以我们必须要使用 root 的身份工作。 但我只想要进行一次该指令而已，此时就使用类似上面的语法吧！好，那接下来，如果我是 root 或者是其他人， 想要变更成为某些特殊帐号，可以使用如下的方法来切换喔！

```
范例四：原本是 dmtsai 这个使用者，想要变换身份成为 vbird1 时？
[dmtsai@study ~]$ su -l vbird1
Password: &lt;==这里输入 vbird1 的密码喔！
[vbird1@study ~]$ su -
Password: &lt;==这里输入 root 的密码喔！
[root@study ~]# id sshd
uid=74（sshd） gid=74（sshd） groups=74（sshd） ... &lt;==确实有存在此人
[root@study ~]# su -l sshd
This account is currently not available.      &lt;==竟然说此人无法切换？
[root@study ~]# finger sshd
Login: sshd                             Name: Privilege-separated SSH
Directory: /var/empty/sshd              Shell: /sbin/nologin
[root@study ~]# exit    &lt;==离开第二次的 su
[vbird1@study ~]$ exit  &lt;==离开第一次的 su
[dmtsai@study ~]$ exit  &lt;==这才是最初的环境！
```

su 就这样简单的介绍完毕，总结一下他的用法是这样的：

- 若要完整的切换到新使用者的环境，必须要使用“ su - username ”或“ su -l username ”，才会连同 PATH/USER/MAIL 等变量都转成新使用者的环境；
- 如果仅想要执行一次 root 的指令，可以利用“ su - -c "指令串" ”的方式来处理；
- 使用 root 切换成为任何使用者时，并不需要输入新使用者的密码；

虽然使用 su 很方便啦，不过缺点是，当我的主机是多人共管的环境时，如果大家都要使用 su 来切换成为 root 的身份，那么不就每个人都得要知道 root 的密码，这样密码太多人知道可能会流出去， 很不妥当呢！怎办？通过 sudo 来处理即可！

13.4.2 sudo

相对于 su 需要了解新切换的使用者密码 （常常是需要 root 的密码）， sudo 的执行则仅需要自己的密码即可！ 甚至可以设置不需要密码即可执行 sudo 呢！由于 sudo 可以让你以其他用户的身份执行指令 （通常是使用 root 的身份来执行指令），因此并非所有人都能够执行 sudo ， 而是仅有规范到 /etc/sudoers 内的用户才能够执行 sudo 这个指令喔！说的这么神奇，下面就来瞧瞧那 sudo 如何使用？



Tips 事实上，一般用户能够具有 sudo 的使用权，就是管理员事先审核通过后，才开放 sudo 的使用权的！因此，除非是信任用户，否则一般用户默认是不能操作 sudo 的喔！

- sudo 的指令用法

由于一开始系统默认仅有 root 可以执行 sudo ，因此下面的范例我们先以 root 的身份来执行，等到谈到 visudo 时，再以一般使用者来讨论其他 sudo 的用法吧！ sudo 的语法如下：



Tips 还记得在安装 CentOS 7 的第三章时，在设置一般帐号的项目中，有个“让这位使用者成为管理员”的选项吧？如果你有勾选该选项的话， 那除了 root 之外，该一般用户确实是可以使用 sudo 的喔（以鸟哥的例子来说， dmtsai 默认竟然可以使用 sudo 了！）！这是因为创建帐号的时候，默认将此用户加入 sudo 的支持中了！详情本章稍后告知！

```
[root@study ~]# sudo [-b] [-u 新使用者帐号]
选项与参数：
-b  ：将后续的指令放到背景中让系统自行执行，而不与目前的 shell 产生影响
-u  ：后面可以接欲切换的使用者，若无此项则代表切换身份为 root 。

范例一：你想要以 sshd 的身份在 /tmp 下面创建一个名为 mysshd 的文件
[root@study ~]# sudo -u sshd touch /tmp/mysshd
[root@study ~]# ll /tmp/mysshd
-rw-r--r--. 1 sshd sshd 0 Jul 21 23:37 /tmp/mysshd
# 特别留意，这个文件的权限是由 sshd 所创建的情况喔！

范例二：你想要以 vbird1 的身份创建 ~vbird1/www 并于其中创建 index.html 文件
[root@study ~]# sudo -u vbird1 sh -c &lt;u&gt;"mkdir ~vbird1/www; cd ~vbird1/www; \&lt;/u
&gt;  &lt;u&gt;echo 'This is index.html file' &gt; index.html"&lt;/u&gt;
[root@study ~]# ll -a ~vbird1/www
drwxr-xr-x. 2 vbird1 vbird1   23 Jul 21 23:38 .
drwx------. 6 vbird1 vbird1 4096 Jul 21 23:38 ..
-rw-r--r--. 1 vbird1 vbird1   24 Jul 21 23:38 index.html
# 要注意，创建者的身份是 vbird1 ，且我们使用 sh -c "一串指令" 来执行的！
```

sudo 可以让你切换身份来进行某项任务，例如上面的两个范例。范例一中，我们的 root 使用 sshd 的权限去进行某项任务！ 要注意，因为我们无法使用“ su - sshd ”去切换系统帐号 （因为系统帐号的 shell 是 /sbin/nologin）， 这个时候 sudo 真是他 X 的好用了！立刻以 sshd 的权限在 /tmp 下面创建文件！查阅一下文件权限你就了解意义啦！ 至于范例二则更使用多重指令串 （通过分号 ; 来延续指令进行），使用 sh -c 的方法来执行一连串的指令， 如此真是好方便！

但是 sudo 默认仅有 root 能使用啊！为什么呢？因为 sudo 的执行是这样的流程：

1. 当使用者执行 sudo 时，系统于 /etc/sudoers 文件中搜寻该使用者是否有执行 sudo 的权限；
2. 若使用者具有可执行 sudo 的权限后，便让使用者“输入使用者自己的密码”来确认；
3. 若密码输入成功，便开始进行 sudo 后续接的指令（但 root 执行 sudo 时，不需要输入密码）；
4. 若欲切换的身份与执行者身份相同，那也不需要输入密码。

所以说，sudo 执行的重点是：“能否使用 sudo 必须要看 /etc/sudoers 的设置值， 而可使用 sudo 者是通过输入使用者自己的密码来执行后续的指令串”喔！由于能否使用与 /etc/sudoers 有关， 所以我们当然要去编辑 sudoers 文件啦！不过，因为该文件的内容是有一定的规范的，因此直接使用 vi 去编辑是不好的。 此时，我们得要通过 visudo 去修改这个文件喔！

- visudo 与 /etc/sudoers

从上面的说明我们可以知道，除了 root 之外的其他帐号，若想要使用 sudo 执行属于 root 的权限指令，则 root 需要先使用 visudo 去修改 /etc/sudoers ，让该帐号能够使用全部或部分的 root 指令功能。为什么要使用 visudo 呢？这是因为 /etc/sudoers 是有设置语法的，如果设置错误那会造成无法使用 sudo 指令的不良后果。因此才会使用 visudo 去修改， 并在结束离开修改画面时，系统会去检验 /etc/sudoers 的语法就是了。

一般来说，visudo 的设置方式有几种简单的方法喔，下面我们以几个简单的例子来分别说明：

- I. 单一使用者可进行 root 所有指令，与 sudoers 文件语法：

假如我们要让 vbird1 这个帐号可以使用 root 的任何指令，基本上有两种作法，第一种是直接通过修改 /etc/sudoers ，方法如下：

```
[root@study ~]# visudo
....（前面省略）....
root    ALL=（ALL）      ALL  &lt;==找到这一行，大约在 98 行左右
vbird1  ALL=（ALL）      ALL &lt;==这一行是你要新增的！
....（下面省略）....
```

有趣吧！其实 visudo 只是利用 vi 将 /etc/sudoers 文件调用出来进行修改而已，所以这个文件就是 /etc/sudoers 啦！ 这个文件的设置其实很简单，如上面所示，如果你找到 98 行 （有 root 设置的那行） 左右，看到的数据就是：

```
使用者帐号  登陆者的来源主机名称=（可切换的身份）  可下达的指令
root                         ALL=（ALL）          ALL   &lt;==这是默认值
```

上面这一行的四个元件意义是：

1. “使用者帐号”：系统的哪个帐号可以使用 sudo 这个指令的意思；

2. “登陆者的来源主机名称”：当这个帐号由哪部主机连线到本 Linux 主机，意思是这个帐号可能是由哪一部网络主机连线过来的， 这个设置值可以指定用户端计算机（信任的来源的意思）。默认值 root 可来自任何一部网络主机
3. “（可切换的身份）”：这个帐号可以切换成什么身份来下达后续的指令，默认 root 可以切换成任何人；
4. “可下达的指令”：可用该身份下达什么指令？这个指令请务必使用绝对路径撰写。 默认 root 可以切换任何身份且进行任何指令之意。

那个 ALL 是特殊的关键字，代表任何身份、主机或指令的意思。所以，我想让 vbird1 可以进行任何身份的任何指令， 就如同上表特殊字体写的那样，其实就是复制上述默认值那一行，再将 root 改成 vbird1 即可啊！ 此时“vbird1 不论来自哪部主机登陆，他可以变换身份成为任何人，且可以进行系统上面的任何指令”之意。 修改完请储存后离开 vi，并以 vbird1 登陆系统后，进行如下的测试看看：

```
[vbird1@study ~]$ tail -n 1 /etc/shadow  &lt;==注意！身份是 vbird1
tail: cannot open `/etc/shadow' for reading: Permission denied
# 因为不是 root 嘛！所以当然不能查询 /etc/shadow

[vbird1@study ~]$ sudo tail -n 1 /etc/shadow &lt;==通过 sudo

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

    #1） Respect the privacy of others.  &lt;==这里仅是一些说明与警示项目
    #2） Think before you type.
    #3） With great power comes great responsibility.

[sudo] password for vbird1: &lt;==注意啊！这里输入的是“ vbird1 自己的密码 ”
pro3:$6$DMilzaKr$OeHeTDQPHzDOz/u5Cyhq1Q1dy...:16636:0:99999:7:::
# 看！vbird1 竟然可以查询 shadow ！
```

注意到了吧！vbird1 输入自己的密码就能够执行 root 的指令！所以，系统管理员当然要了解 vbird1 这个用户的“操守”才行！否则随便设置一个使用者，他恶搞系统怎办？另外，一个一个设置太麻烦了， 能不能使用群组的方式来设置呢？参考下面的第二种方式吧。

- II. 利用 wheel 群组以及免密码的功能处理 visudo

我们在本章前面曾经创建过 pro1, pro2, pro3 ，这三个用户能否通过群组的功能让这三个人可以管理系统？ 可以的，而且很简单！同样我们使用实际案例来说明：

```
[root@study ~]# visudo  &lt;==同样的，请使用 root 先设置
....（前面省略）....
%wheel     ALL=（ALL）     ALL &lt;==大约在 106 行左右，请将这行的 # 拿掉！
# 在最左边加上 % ，代表后面接的是一个“群组”之意！改完请储存后离开

[root@study ~]# usermod -a -G wheel pro1 &lt;==将 pro1 加入 wheel 的支持
```

上面的设置值会造成“任何加入 wheel 这个群组的使用者，就能够使用 sudo 切换任何身份来操作任何指令”的意思。 你当然可以将 wheel 换成你自己想要的群组名。接下来，请分别切换身份成为 pro1 及 pro2 试看看 sudo 的运行。

```
[pro1@study ~]$ sudo tail -n 1 /etc/shadow &lt;==注意身份是 pro1
....（前面省略）....
[sudo] password for pro1:  &lt;==输入 pro1 的密码喔！
pro3:$6$DMilzaKr$OeHeTDQPHzDOz/u5Cyhq1Q1dy...:16636:0:99999:7:::

[pro2@study ~]$ sudo tail -n 1 /etc/shadow &lt;==注意身份是 pro2
[sudo] password for pro2:  &lt;==输入 pro2 的密码喔！
pro2 is not in the sudoers file.  This incident will be reported.
# 仔细看错误讯息他是说这个 pro2 不在 /etc/sudoers 的设置中！
```

这样理解群组了吧？如果你想要让 pro3 也支持这个 sudo 的话，不需要重新使用 visudo ，只要利用 usermod 去修改 pro3 的群组支持，让 pro3 用户加入 wheel 群组当中，那他就能够进行 sudo 啰！ 好了！那么现在你知道为啥在安装时创建的用户，就是那个 dmstai 默认可以使用 sudo 了吗？请使用“ id dmtsai ”看看， 这个用户是否有加入 wheel 群组呢？嘿嘿！了解乎？



Tips 从 CentOS 7 开始，在 sudoers 文件中，默认已经开放 %wheel 那一行啰！以前的 CentOS 旧版本都是没有启用的呢！

简单吧！不过，既然我们都信任这些 sudo 的用户了，能否提供“不需要密码即可使用 sudo ”呢？ 就通过如下的方式：

```
[root@study ~]# visudo  &lt;==同样的，请使用 root 先设置
....（前面省略）....
%wheel     ALL=（ALL）    NOPASSWD: ALL &lt;==大约在 109 行左右，请将 # 拿掉！
# 在最左边加上 % ，代表后面接的是一个“群组”之意！改完请储存后离开
```

重点是那个 NOPASSWD 啦！该关键字是免除密码输入的意思喔！

- III. 有限制的指令操作：

上面两点都会让使用者能够利用 root 的身份进行任何事情！这样总是不太好～如果我想要让使用者仅能够进行部分系统任务， 比方说，系统上面的 myuser1 仅能够帮 root 修改其他使用者的密码时，亦即“当使用者仅能使用 passwd 这个指令帮忙 root 修改其他用户的密码”时，你该如何撰写呢？可以这样做：

```
[root@study ~]# visudo  &lt;==注意是 root 身份
myuser1      ALL=（root）    /usr/bin/passwd  &lt;==最后指令务必用绝对路径
```

上面的设置值指的是“myuser1 可以切换成为 root 使用 passwd 这个指令”的意思。其中要注意的是： 指令字段必须要填写绝对路径才行！否则 visudo 会出现语法错误的状况发生！ 此外，上面的设置是有问题的！我们使用下面的指令操作来让您了解：

```
[myuser1@study ~]$ sudo passwd myuser3  &lt;==注意，身份是 myuser1
[sudo] password for myuser1:  &lt;==输入 myuser1 的密码
Changing password for user myuser3\. &lt;==下面改的是 myuser3 的密码喔！这样是正确的
New password:
Retype new password:
passwd: all authentication tokens updated successfully.

[myuser1@study ~]$ sudo passwd
Changing password for user root.  &lt;==见鬼！怎么会去改 root 的密码？
```

恐怖啊！我们竟然让 root 的密码被 myuser1 给改变了！下次 root 回来竟无法登陆系统...欲哭无泪～怎办？ 所以我们必须要限制使用者的指令参数！修改的方法为将上述的那行改一改先：

```
[root@study ~]# visudo  &lt;==注意是 root 身份
myuser1    ALL=（root）  !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]*, !/usr/bin/passwd roo
```

在设置值中加上惊叹号“！”代表“不可执行”的意思。因此上面这一行会变成：可以执行“passwd 任意字符”，但是“ passwd ”与“ passwd root ”这两个指令例外！ 如此一来 myuser1 就无法改变 root 的密码了！这样这位使用者可以具有 root 的能力帮助你修改其他用户的密码，而且也不能随意改变 root 的密码！很有用处的！

- IV. 通过别名创建 visudo：

如上述第三点，如果我有 15 个用户需要加入刚刚的管理员行列，那么我是否要将上述那长长的设置写入 15 行啊？ 而且如果想要修改命令或者是新增命令时，那我每行都需要重新设置，很麻烦ㄟ！有没有更简单的方式？ 是有的！通过别名即可！我们 visudo 的别名可以是“指令别名、帐号别名、主机别名”等。不过这里我们仅介绍帐号别名， 其他的设置值有兴趣的话，可以自行玩玩！

假设我的 pro1, pro2, pro3 与 myuser1, myuser2 要加入上述的密码管理员的 sudo 列表中，那我可以创立一个帐号别名称为 ADMPW 的名称，然后将这个名称处理一下即可。处理的方式如下：

```
[root@study ~]# visudo  &lt;==注意是 root 身份
User_Alias ADMPW = pro1, pro2, pro3, myuser1, myuser2
Cmnd_Alias ADMPWCOM = !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]*, !/usr/bin/passwd root
ADMPW   ALL=（root）  ADMPWCOM
```

我通过 User_Alias 创建出一个新帐号，这个帐号名称一定要使用大写字符来处理，包括 Cmnd_Alias（命令别名）、Host_Alias（来源主机名称别名） 都需要使用大写字符的！这个 ADMPW 代表后面接的那些实际帐号。 而该帐号能够进行的指令就如同 ADMPWCOM 后面所指定的那样！上表最后一行则写入这两个别名 （帐号与指令别名）， 未来要修改时，我只要修改 User_Alias 以及 Cmnd_Alias 这两行即可！设置方面会比较简单有弹性喔！

- V. sudo 的时间间隔问题：

或许您已经发现了，那就是，如果我使用同一个帐号在短时间内重复操作 sudo 来运行指令的话， 在第二次执行 sudo 时，并不需要输入自己的密码！sudo 还是会正确的运行喔！为什么呢？ 第一次执行 sudo 需要输入密码，是担心由于使用者暂时离开座位，但有人跑来你的座位使用你的帐号操作系统之故。 所以需要你输入一次密码重新确认一次身份。

两次执行 sudo 的间隔在五分钟内，那么再次执行 sudo 时就不需要再次输入密码了， 这是因为系统相信你在五分钟内不会离开你的作业，所以执行 sudo 的是同一个人！呼呼！真是很人性化的设计啊～ ^_^。不过如果两次 sudo 操作的间隔超过 5 分钟，那就得要重新输入一次你的密码了 [4]

- VI. sudo 搭配 su 的使用方式：

很多时候我们需要大量执行很多 root 的工作，所以一直使用 sudo 觉得很烦乁！那有没有办法使用 sudo 搭配 su ， 一口气将身份转为 root ，而且还用使用者自己的密码来变成 root 呢？是有的！而且方法简单的会让你想笑！ 我们创建一个 ADMINS 帐号别名，然后这样做：

```
[root@study ~]# visudo
User_Alias  ADMINS = pro1, pro2, pro3, myuser1
ADMINS ALL=（root）  /bin/su -
```

接下来，上述的 pro1, pro2, pro3, myuser1 这四个人，只要输入“ sudo su - ”并且输入“自己的密码”后， 立刻变成 root 的身份！不但 root 密码不会外流，使用者的管理也变的非常方便！ 这也是实务上面多人共管一部主机时常常使用的技巧呢！这样管理确实方便，不过还是要强调一下大前提， 那就是“这些你加入的使用者，全部都是你能够信任的用户”！

13.5 使用者的特殊 shell 与 PAM 模块

我们前面一直谈到的大多是一般身份使用者与系统管理员 （root） 的相关操作， 而且大多是讨论关于可登陆系统的帐号来说。那么换个角度想，如果我今天想要创建的， 是一个“仅能使用 mail server 相关邮件服务的帐号，而该帐号并不能登陆 Linux 主机”呢？如果不能给予该帐号一个密码，那么该帐号就无法使用系统的各项资源，当然也包括 mail 的资源， 而如果给予一个密码，那么该帐号就可能可以登陆 Linux 主机啊！呵呵～伤脑筋吧～ 所以，下面让我们来谈一谈这些有趣的话题啰！

另外，在本章之前谈到过 /etc/login.defs 文件中，关于密码长度应该默认是 5 个字串长度，但是我们上面也谈到，该设置值已经被 PAM 模块所取代了，那么 PAM 是什么？为什么他可以影响我们使用者的登陆呢？这里也要来谈谈的！

13.5.1 特殊的 shell, /sbin/nologin

在本章一开头的 passwd 文件结构里面我们就谈过系统帐号这玩意儿，这玩意儿的 shell 就是使用 /sbin/nologin ，重点在于系统帐号是不需要登陆的！所以我们就给他这个无法登陆的合法 shell。 使用了这个 shell 的用户即使有了密码，你想要登陆时他也无法登陆，因为会出现如下的讯息喔：

```
This account is currently not available.
```

我们所谓的“无法登陆”指的仅是：“这个使用者无法使用 bash 或其他 shell 来登陆系统”而已，并不是说这个帐号就无法使用其他的系统资源喔！ 举例来说，各个系统帐号，打印工作由 lp 这个帐号在管理， WWW 服务由 apache 这个帐号在管理， 他们都可以进行系统程序的工作，但是“就是无法登陆主机取得互动的 shell”而已啦！^_^

换个角度来想，如果我的 Linux 主机提供的是邮件服务，所以说，在这部 Linux 主机上面的帐号， 其实大部分都是用来收受主机的信件而已，并不需要登陆主机的呢！ 这个时候，我们就可以考虑让单纯使用 mail 的帐号以 /sbin/nologin 做为他们的 shell ， 这样，最起码当我的主机被尝试想要登陆系统以取得 shell 环境时，可以拒绝该帐号呢！

另外，如果我想要让某个具有 /sbin/nologin 的使用者知道，他们不能登陆主机时， 其实我可以创建“ /etc/nologin.txt ”这个文件， 并且在这个文件内说明不能登陆的原因，那么下次当这个使用者想要登陆系统时， 屏幕上出现的就会是 /etc/nologin.txt 这个文件的内容，而不是默认的内容了！

例题：当使用者尝试利用纯 mail 帐号 （例如 myuser3） 时，利用 /etc/nologin.txt 告知使用者不要利用该帐号登陆系统。答：直接以 vim 编辑该文件，内容可以是这样：

```
[root@study ~]# vim /etc/nologin.txt
This account is system account or mail account.
Please DO NOT use this account to login my Linux server.
```

想要测试时，可以使用 myuser3 （此帐号的 shell 是 /sbin/nologin） 来测试看看！

```
[root@study ~]# su - myuser3
This account is system account or mail account.
Please DO NOT use this account to login my Linux server.
```

结果会发现与原本的默认讯息不一样喔！ ^_^

13.5.2 PAM 模块简介

在过去，我们想要对一个使用者进行认证 （authentication），得要要求使用者输入帐号密码， 然后通过自行撰写的程序来判断该帐号密码是否正确。也因为如此，我们常常得使用不同的机制来判断帐号密码， 所以搞的一部主机上面拥有多个各别的认证系统，也造成帐号密码可能不同步的验证问题！ 为了解决这个问题因此有了 PAM （Pluggable Authentication Modules, 嵌入式模块） 的机制！

PAM 可以说是一套应用程序接口 （Application Programming Interface, API），他提供了一连串的验证机制，只要使用者将验证阶段的需求告知 PAM 后， PAM 就能够回报使用者验证的结果 （成功或失败）。由于 PAM 仅是一套验证的机制，又可以提供给其他程序所调用引用，因此不论你使用什么程序，都可以使用 PAM 来进行验证，如此一来，就能够让帐号密码或者是其他方式的验证具有一致的结果！也让程序设计师方便处理验证的问题喔！ [5]

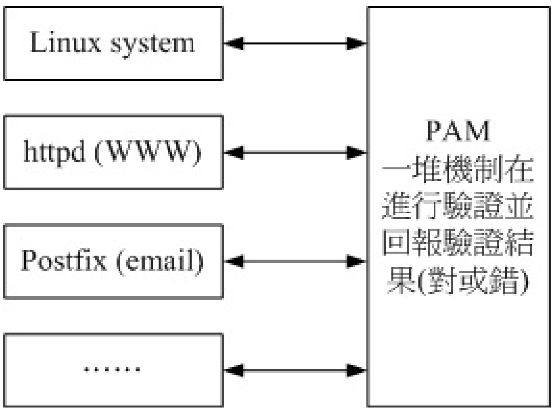


图13.5.1、PAM 模块与其他程序的相关性

如上述的图示， PAM 是一个独立的 API 存在，只要任何程序有需求时，可以向 PAM 发出验证要求的通知， PAM 经过一连串的验证后，将验证的结果回报给该程序，然后该程序就能够利用验证的结果来进行可登陆或显示其他无法使用的讯息。 这也就是说，你可以在写程序的时候将 PAM 模块的功能加入，就能够利用 PAM 的验证功能啰。 因此目前很多程序都会利用 PAM 喔！所以我们才要来学习他啊！

PAM 用来进行验证的数据称为模块 （Modules），每个 PAM 模块的功能都不太相同。举例来说， 还记得我们在本章使用 passwd 指令时，如果随便输入字典上面找的到的字串，passwd 就会回报错误信息了！这是为什么呢？这就是 PAM 的 pam_cracklib.so 模块的功能！他能够判断该密码是否在字典里面！ 并回报给密码修改程序，此时就能够了解你的密码强度了。

所以，当你有任何需要判断是否在字典当中的密码字串时，就可以使用 pam_cracklib.so 这个模块来验证！ 并根据验证的回报结果来撰写你的程序呢！这样说，可以理解 PAM 的功能了吧？

13.5.3 PAM 模块设置语法

PAM 借由一个与程序相同文件名的配置文件来进行一连串的认证分析需求。我们同样以 passwd 这个指令的调用 PAM 来说明好了。 当你执行 passwd 后，这支程序调用 PAM 的流程是：

1. 使用者开始执行 /usr/bin/passwd 这支程序，并输入密码；
2. passwd 调用 PAM 模块进行验证；
3. PAM 模块会到 /etc/pam.d/ 找寻与程序 （passwd） 同名的配置文件；
4. 依据 /etc/pam.d/passwd 内的设置，引用相关的 PAM 模块逐步进行验证分析；
5. 将验证结果 （成功、失败以及其他讯息） 回传给 passwd 这支程序；
6. passwd 这支程序会根据 PAM 回传的结果决定下一个动作 （重新输入新密码或者通过验证！）

从上头的说明，我们会知道重点其实是 /etc/pam.d/ 里面的配置文件，以及配置文件所调用的 PAM 模块进行的验证工作！ 既然一直谈到 passwd 这个密码修改指令，那我们就来看看 /etc/pam.d/passwd 这个配置文件的内容是怎样吧！

```
[root@study ~]# cat /etc/pam.d/passwd
#%PAM-1.0  &lt;==PAM版本的说明而已！
auth        include       system-auth   &lt;==每一行都是一个验证的过程
account     include       system-auth
password    substack      system-auth
-password    optional     pam_gnome_keyring.so use_authtok
password    substack      postlogin
验证类别     控制标准       PAM 模块与该模块的参数
```

在这个配置文件当中，除了第一行宣告 PAM 版本之外，其他任何“ # ”开头的都是注解，而每一行都是一个独立的验证流程， 每一行可以区分为三个字段，分别是验证类别（type）、控制标准（flag）、PAM的模块与该模块的参数。 下面我们先来谈谈验证类别与控制标准这两项数据吧！



Tips 你会发现在我们上面的表格当中出现的是“ include （包括） ”这个关键字，他代表的是“请调用后面的文件来作为这个类别的验证”， 所以，上述的每一行都要重复调用 /etc/pam.d/system-auth 那个文件来进行验证的意思！

- 第一个字段：验证类别 （Type）

验证类别主要分为四种，分别说明如下：

- auth 是 authentication （认证） 的缩写，所以这种类别主要用来检验使用者的身份验证，这种类别通常是需要密码来检验的， 所以后续接的模块是用来检验使用者的身份。
- account account （帐号） 则大部分是在进行 authorization （授权），这种类别则主要在检验使用者是否具有正确的使用权限， 举例来说，当你使用一个过期的密码来登陆时，当然就无法正确的登陆了。
- session session 是会议期间的意思，所以 session 管理的就是使用者在这次登陆 （或使用这个指令） 期间，PAM 所给予的环境设置。 这个类别通常用在记录使用者登陆与登出时的信息！例如，如果你常常使用 su 或者是 sudo 指令的话， 那么应该可以在 /var/log/secure 里面发现很多关于 pam 的说明，而且记载的数据是“session open, session close”的信息！
- password password 就是密码嘛！所以这种类别主要在提供验证的修订工作，举例来说，就是修改/变更密码啦！

这四个验证的类型通常是有顺序的，不过也有例外就是了。 会有顺序的原因是，（1）我们总是得要先验证身份 （auth） 后， （2）系统才能够借由使用者的身份给予适当的授权与权限设置 （account），而且（3）登陆与登出期间的环境才需要设置， 也才需要记录登陆与登出的信息 （session）。如果在运行期间需要密码修订时，（4）才给予 password 的类别。这样说起来， 自然是需要有点顺序吧！

- 第二个字段：验证的控制旗标 （control flag）

那么“验证的控制旗标（control flag）”又是什么？简单的说，他就是“验证通过的标准”啦！ 这个字段在管控该验证的放行方式，主要也分为四种控制方式：

- required 此验证若成功则带有 success （成功） 的标志，若失败则带有 failure 的标志，但不论成功或失败都会继续后续的验证流程。 由于后续的验证流程可以继续进行，因此相当有利于数据的登录 （log） ，这也是 PAM 最常使用 required 的原因。

- requisite 若验证失败则立刻回报原程序 failure 的标志，并终止后续的验证流程。若验证成功则带有 success 的标志并继续后续的验证流程。 这个项目与 required 最大的差异，就在于失败的时候还要不要继续验证下去？由于 requisite 是失败就终止， 因此失败时所产生的 PAM 信息就无法通过后续的模块来记录了。
- sufficient 若验证成功则立刻回传 success 给原程序，并终止后续的验证流程；若验证失败则带有 failure 标志并继续后续的验证流程。 这玩意儿与 requisits 刚好相反！
- optional 这个模块控制项目大多是在显示讯息而已，并不是用在验证方面的。

如果将这些控制旗标以图示的方式配合成功与否的条件绘图，会有点像下面这样：

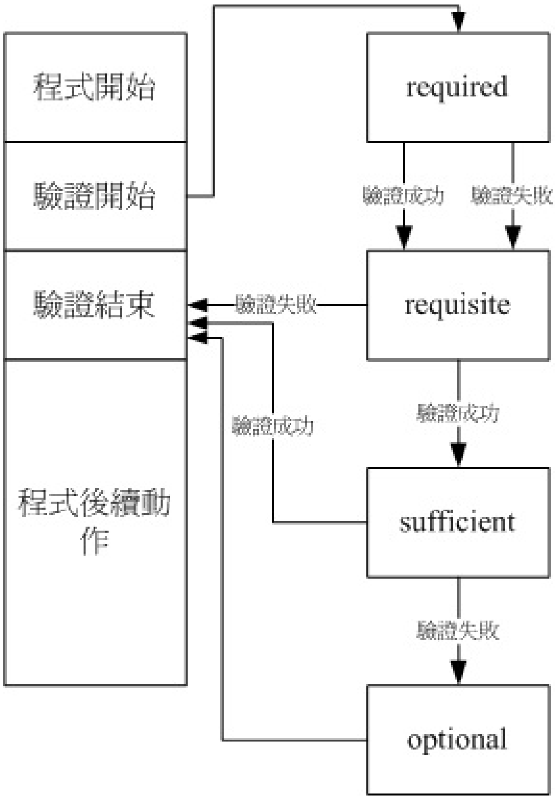


图13.5.2、PAM 控制旗标所造成的回报流程

程序运行过程中遇到验证时才会去调用 PAM ，而 PAM 验证又分很多类型与控制，不同的控制旗标所回报的讯息并不相同。 如上图所示， requisite 失败就回报了并不会继续，而 sufficient 则是成功就回报了也不会继续。 至于验证结束后所回报的信息通常是“succes 或 failure ”而已，后续的流程还需要该程序的判断来继续执行才行。

13.5.4 常用模块简介

谈完了配置文件的语法后，现在让我们来查阅一下 CentOS 5.x 提供的 PAM 默认文件的内容是啥吧！ 由于我们常常需要通过各种方式登陆 （login） 系统，因此就来看看登陆所需要的 PAM 流程为何：

```
[root@study ~]# cat /etc/pam.d/login
#%PAM-1.0
auth [user_unknown=ignore success=ok ignore=ignore default=bad] pam_securetty.so
auth       substack     system-auth
auth       include      postlogin
account    required     pam_nologin.so
account    include      system-auth
password   include      system-auth
# pam_selinux.so close should be the first session rule
session    required     pam_selinux.so close
session    required     pam_loginuid.so
session    optional     pam_console.so
# pam_selinux.so open should only be followed by sessions to be executed in the user cont
session    required     pam_selinux.so open
session    required     pam_namespace.so
session    optional     pam_keyinit.so force revoke
session    include      system-auth
session    include      postlogin
-session   optional     pam_ck_connector.so
# 我们可以看到，其实 login 也调用多次的 system-auth ，所以下面列出该配置文件

[root@study ~]# cat /etc/pam.d/system-auth
#%PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth        required      pam_env.so
auth        sufficient    pam_fprintd.so
auth        sufficient    pam_unix.so nullok try_first_pass
auth        requisite     pam_succeed_if.so uid &gt;= 1000 quiet_success
auth        required      pam_deny.so

account     required      pam_unix.so
account     sufficient    pam_localuser.so
account     sufficient    pam_succeed_if.so uid &lt; 1000 quiet
account     required      pam_permit.so

password    requisite     pam_pwquality.so try_first_pass local_users_only retry=3 authto
password    sufficient    pam_unix.so sha512 shadow nullok try_first_pass use_authtok
password    required      pam_deny.so

session     optional      pam_keyinit.so revoke
session     required      pam_limits.so
-session     optional      pam_systemd.so
session     [success=1 default=ignore] pam_succeed_if.so service in crond quiet use_uid
session     required      pam_unix.so
```

上面这个表格当中使用到非常多的 PAM 模块，每个模块的功能都不太相同，详细的模块情报可以在你的系统中找到：

- /etc/pam.d/*：每个程序个别的 PAM 配置文件；
- /lib64/security/*：PAM 模块文件的实际放置目录；
- /etc/security/*：其他 PAM 环境的配置文件；
- /usr/share/doc/pam-*/：详细的 PAM 说明文档。

例如鸟哥使用未 update 过的 CentOS 7.1 ，pam*nologin* 说明文档在： */usr/share/doc/pam-1.1.8/txts/README.pam_nologin*。你可以自行查阅一下该模块的功能。 鸟哥这里仅简单介绍几个较常使用的模块，详细的信息还得要您努力查阅参考书呢！ ^^

- pam_securetty.so： 限制系统管理员 （root） 只能够从安全的 （secure） 终端机登陆；那什么是终端机？例如 tty1, tty2 等就是传统的终端机设备名称。那么安全的终端机设置呢？ 就写在 /etc/securetty 这个文件中。你可以查阅一下该文件， 就知道为什么 root 可以从 tty1~tty7 登陆，但却无法通过 telnet 登陆 Linux 主机了！
- pam_nologin.so： 这个模块可以限制一般使用者是否能够登陆主机之用。当 /etc/nologin 这个文件存在时，则所有一般使用者均无法再登陆系统了！若 /etc/nologin 存在，则一般使用者在登陆时， 在他们的终端机上会将该文件的内容显示出来！所以，正常的情况下，这个文件应该是不能存在系统中的。 但这个模块对 root 以及已经登陆系统中的一般帐号并没有影响。 （注意喔！这与 /etc/nologin.txt 并不相同！）
- pam_selinux.so： SELinux 是个针对程序来进行细部管理权限的功能，SELinux 这玩意儿我们会在第十六章的时候再来详细谈论。由于 SELinux 会影响到使用者执行程序的权限，因此我们利用 PAM 模块，将 SELinux 暂时关闭，等到验证通过后， 再予以启动！
- pam_console.so： 当系统出现某些问题，或者是某些时刻你需要使用特殊的终端接口 （例如 RS232 之类的终端连线设备） 登陆主机时， 这个模块可以帮助处理一些文件权限的问题，让使用者可以通过特殊终端接口 （console） 顺利的登陆系统。
- pam_loginuid.so： 我们知道系统帐号与一般帐号的 UID 是不同的！一般帐号 UID 均大于 1000 才合理。 因此，为了验证使用者的 UID 真的是我们所需要的数值，可以使用这个模块来进行规范！
- pam_env.so： 用来设置环境变量的一个模块，如果你有需要额外的环境变量设置，可以参考 /etc/security/pam_env.conf 这个文件的详细说明。
- pam_unix.so： 这是个很复杂且重要的模块，这个模块可以用在验证阶段的认证功能，可以用在授权阶段的帐号授权管理， 可以用在会议阶段的登录文件记录等，甚至也可以用在密码更新阶段的检验！非常丰富的功能！ 这个模块在早期使用得相当频繁喔！
- pam_pwquality.so： 可以用来检验密码的强度！包括密码是否在字典中，密码输入几次都失败就断掉此次连线等功能，都是这模块提供的！ 最早之前其实使用的是 pam_cracklib.so 这个模块，后来改成 pam_pwquality.so 这个模块，但此模块完全相容于 pam_cracklib.so， 同时提供了 /etc/security/pwquality.conf 这个文件可以额外指定默认值！比较容易处理修改！
- pam_limits.so： 还记得我们在第十章谈到的 ulimit 吗？ 其实那就是这个模块提供的能力！还有更多细部的设置可以参考： /etc/security/limits.conf 内的说明。

了解了这些模块的大致功能后，言归正传，讨论一下 login 的 PAM 验证机制流程是这样的：

1. 验证阶段 （auth） ：首先，（a）会先经过 pam_securetty.so 判断，如果使用者是 root 时，则会参考 /etc/securetty 的设置； 接下来（b）经过 pam_env.so 设置额外的环境变量；再（c）通过 pam_unix.so 检验密码，若通过则回报 login 程序；若不通过则（d）

继续往下以 pam_succeed_if.so 判断 UID 是否大于 1000 ，若小于 1000则回报失败，否则再往下（e）以 pam_deny.so 拒绝连线。

2. 授权阶段（account）：（a）先以 pam_nologin.so 判断 /etc/nologin 是否存在，若存在则不许一般使用者登陆；（b）接下来以 pam_unix.so 及 pam_localuser.so 进行帐号管理，再以（c） pam_succeed_if.so 判断 UID 是否小于 1000 ，若小于 1000 则不记录登录信息。（d）最后以 pam_permit.so 允许该帐号登陆。

3. 密码阶段（password）：（a）先以 pam_pwquality.so 设置密码仅能尝试错误 3 次；（b）接下来以 pam_unix.so 通过 sha512, shadow 等功能进行密码检验，若通过则回报 login 程序，若不通过则（c）以 pam_deny.so 拒绝登陆。

4. 会议阶段（session）：（a）先以 pam_selinux.so 暂时关闭 SELinux；（b）使用 pam_limits.so 设置好使用者能够操作的系统资源；（c）登陆成功后开始记录相关信息在登录文件中；（d）以 pam_loginuid.so 规范不同的 UID 权限；（e）打开 pam_selinux.so 的功能。

总之，就是依据验证类别（type）来看，然后先由 login 的设置值去查阅，如果出现“ include system-auth ” 就转到 system-auth 文件中的相同类别，去取得额外的验证流程就是了。然后再到下一个验证类别，最终将所有的验证跑完！ 就结束这次的 PAM 验证啦！

经过这样的验证流程，现在你知道为啥 /etc/nologin 存在会有问题，也会知道为何你使用一些远端连线机制时， 老是无法使用 root 登陆的问题了吧？没错！这都是 PAM 模块提供的功能啦！

例题：为什么 root 无法以 telnet 直接登陆系统，但是却能够使用 ssh 直接登陆？答：一般来说， telnet 会引用 login 的 PAM 模块，而 login 的验证阶段会有 /etc/securetty 的限制！ 由于远端连线属于 pts/n（n 为数字） 的动态终端机接口设备名称，并没有写入到 /etc/securetty ， 因此 root 无法以 telnet 登陆远端主机。至于 ssh 使用的是 /etc/pam.d/sshd 这个模块， 你可以查阅一下该模块，由于该模块的验证阶段并没有加入 pam_securetty ，因此就没有 /etc/securetty 的限制！故可以从远端直接连线到服务器端。

另外，关于 telnet 与 ssh 的细部说明，请参考鸟哥的 Linux 私房菜服务器篇

13.5.5 其他相关文件

除了前一小节谈到的 /etc/securetty 会影响到 root 可登陆的安全终端机， /etc/nologin 会影响到一般使用者是否能够登陆的功能之外，我们也知道 PAM 相关的配置文件在 /etc/pam.d ，说明文档在 /usr/share/doc/pam-（版本） ，模块实际在 /lib64/security/ 。那么还有没有相关的 PAM 文件呢？ 是有的，主要都在 /etc/security 这个目录内！我们下面介绍几个可能会用到的配置文件喔！

- limits.conf

我们在第十章谈到的 ulimit 功能中， 除了修改使用者的 ~/.bashrc 配置文件之外，其实系统管理员可以统一借由 PAM 来管理的！ 那就是 /etc/security/limits.conf 这个文件的设置了。这个文件的设置很简单，你可以自行参考一下该文件内容。 我们这里仅作个简单的介绍：

```
范例一：vbird1 这个用户只能创建 100MB 的文件，且大于 90MB 会警告
[root@study ~]# vim /etc/security/limits.conf
vbird1     soft         fsize          90000
vbird1     hard         fsize         100000
#帐号     限制依据      限制项目        限制值
# 第一字段为帐号，或者是群组！若为群组则前面需要加上 @ ，例如 @projecta
# 第二字段为限制的依据，是严格（hard），还是仅为警告（soft）；
# 第三字段为相关限制，此例中限制文件大小，
# 第四字段为限制的值，在此例中单位为 KB。
# 若以 vbird1 登陆后，进行如下的操作则会有相关的限制出现！

[vbird1@study ~]$ ulimit -a
....（前面省略）....
file size                 （blocks, -f） 90000
....（后面省略）....

[vbird1@study ~]$ dd if=/dev/zero of=test bs=1M count=110
File size limit exceeded
[vbird1@study ~]$ ll --block-size=K test
-rw-rw-r--. 1 vbird1 vbird1 90000K Jul 22 01:33 test
# 果然有限制到了

范例二：限制 pro1 这个群组，每次仅能有一个使用者登陆系统 （maxlogins）
[root@study ~]# vim /etc/security/limits.conf
@pro1    hard    maxlogins    1
# 如果要使用群组功能的话，这个功能似乎对初始群组才有效喔！而如果你尝试多个 pro1 的登陆时，
# 第二个以后就无法登陆了。而且在 /var/log/secure 文件中还会出现如下的信息：
# pam_limits（login:session）: Too many logins （max 1） for pro1
```

这个文件挺有趣的，而且是设置完成就生效了，你不用重新启动任何服务的！ 但是 PAM 有个特殊的地方，由于他是在程序调用时才予以设置的，因此你修改完成的数据， 对于已登陆系统中的使用者是没有效果的，要等他再次登陆时才会生效喔！另外， 上述的设置请在测试完成后立刻注解掉，否则下次这两个使用者登陆就会发生些许问题啦！ ^_^

- /var/log/secure, /var/log/messages

如果发生任何无法登陆或者是产生一些你无法预期的错误时，由于 PAM 模块都会将数据记载在 /var/log/secure 当中，所以发生了问题请务必到该文件内去查询一下问题点！举例来说，我们在 limits.conf 的介绍内的范例二，就有谈到多重登陆的错误可以到 /var/log/secure 内查阅了！ 这样你也就知道为何第二个 pro1 无法登陆啦！^_^

13.6 Linux 主机上的使用者讯息传递

谈了这么多的帐号问题，总是该要谈一谈，那么如何针对系统上面的使用者进行查询吧？ 想几个状态，如果你在 Linux 上面操作时，刚好有其他的使用者也登陆主机，你想要跟他对谈，该如何是好？ 你想要知道某个帐号的相关信息，该如何查阅？呼呼！下面我们就来聊一聊～

13.6.1 查询使用者： w, who, last, lastlog

如何查询一个使用者的相关数据呢？这还不简单，我们之前就提过了 id, finger 等指令了，都可以让您了解到一个使用者的相关信息啦！那么想要知道使用者到底啥时候登陆呢？ 最简单可以使用 last 检查啊！这个玩意儿我们也在 第十章 bash 提过了， 您可以自行前往参考啊！ 简单的很。



Tips 早期的 Red Hat 系统的版本中， last 仅会列出当月的登陆者信息，不过在我们的 CentOS 5.x 版以后， last 可以列出从系统创建之后到目前为止的所有登陆者信息！这是因为登录文件轮替的设置不同所致。 详细的说明可以参考后续的第十八章登录文件简介。

那如果你想要知道目前已登陆在系统上面的使用者呢？可以通过 w 或 who 来查询喔！如下范例所示：

```
[root@study ~]# w
 01:49:18 up 25 days,  3:34,  3 users,  load average: 0.00, 0.01, 0.05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU WHAT
dmtsai   tty2                      07Jul15 12days  0.03s  0.03s -bash
dmtsai   pts/0    172.16.200.254   00:18    6.00s  0.31s  0.11s sshd: dmtsai [priv]
# 第一行显示目前的时间、开机 （up） 多久，几个使用者在系统上平均负载等；
# 第二行只是各个项目的说明，
# 第三行以后，每行代表一个使用者。如上所示，dmtsai 登陆并取得终端机名 tty2 之意。

[root@study ~]# who
dmtsai   tty2         2015-07-07 23:07
dmtsai   pts/0        2015-07-22 00:18 （192.168.1.100）
```

另外，如果您想要知道每个帐号的最近登陆的时间，则可以使用 lastlog 这个指令喔！ lastlog 会去读取 /var/log/lastlog 文件，结果将数据输出如下表：

```
[root@study ~]# lastlog
Username         Port     From             Latest
root             pts/0                     Wed Jul 22 00:26:08 +0800 2015
bin                                        **Never logged in**
....（中间省略）....
dmtsai           pts/1    192.168.1.100    Wed Jul 22 01:08:07 +0800 2015
vbird1           pts/0                     Wed Jul 22 01:32:17 +0800 2015
pro3                                       **Never logged in**
....（以下省略）....
```

这样就能够知道每个帐号的最近登陆的时间啰～ ^_^

13.6.2 使用者对谈： write, mesg, wall

那么我是否可以跟系统上面的使用者谈天说地呢？当然可以啦！利用 write 这个指令即可。write 可以直接将讯息传给接收者啰！举例来说，我们的 Linux 目前有 vbird1 与 root 两个人在线上， 我的 root 要跟 vbird1 讲话，可以这样做：

```
[root@study ~]# write 使用者帐号 [使用者所在终端接口]

[root@study ~]# who
vbird1   tty3         2015-07-22 01:55  &lt;==有看到 vbird1 在线上
root     tty4         2015-07-22 01:56

[root@study ~]# write vbird1 pts/2
Hello, there:
Please don't do anything wrong...  &lt;==这两行是 root 写的信息！
# 结束时，请按下 [crtl]-d 来结束输入。此时在 vbird1 的画面中，会出现：

Message from root@study.centos.vbird on tty4 at 01:57 ...
Hello, there:
Please don't do anything wrong...
EOF
```

怪怪～立刻会有讯息回应给 vbird1 ！不过......当时 vbird1 正在查数据，哇！ 这些讯息会立刻打断 vbird1 原本的工作喔！所以，如果 vbird1 这个人不想要接受任何讯息，直接下达这个动作：

```
[vbird1@study ~]$ mesg n
[vbird1@study ~]$ mesg
is n
```

不过，这个 mesg 的功能对 root 传送来的讯息没有抵挡的能力！所以如果是 root 传送讯息，vbird1 还是得要收下。 但是如果 root 的 mesg 是 n 的，那么 vbird1 写给 root 的信息会变这样：

```
[vbird1@study ~]$ write root
write: root has messages disabled
```

了解乎？如果想要解开的话，再次下达“ mesg y ”就好啦！想要知道目前的 mesg 状态，直接下达“ mesg ”即可！瞭呼？ 相对于 write 是仅针对一个使用者来传“简讯”，我们还可以“对所有系统上面的使用者传送简讯 （广播）”哩～ 如何下达？用 wall 即可啊！他的语法也是很简单的喔！

```
[root@study ~]# wall "I will shutdown my linux server..."
```

然后你就会发现所有的人都会收到这个简讯呢！连发送者自己也会收到耶！

13.6.3 使用者邮件信箱： mail

使用 wall, write 毕竟要等到使用者在线上才能够进行，有没有其他方式来联络啊？ 不是说每个 Linux 主机上面的使用者都具有一个 mailbox 吗？ 我们可否寄信给使用者啊！呵呵！当然可以啊！我们可以寄、收 mailbox 内的信件呢！ 一般来说， mailbox 都会放置在 /var/spool/mail 里面，一个帐号一个 mailbox （文件）。 举例来说，我的 vbird1 就具有 /var/spool/mail/vbird1 这个 mailbox 喔！

那么我该如何寄出信件呢？就直接使用 mail 这个指令即可！这个指令的用法很简单的，直接这样下达：“ mail -s "邮件标题" username@localhost ”即可！ 一般来说，如果是寄给本机上的使用者，基本上，连“ @localhost ”都不用写啦！ 举例来说，我以 root 寄信给 vbird1 ，信件标题是“ nice to meet you ”，则：

```
[root@study ~]# mail -s "nice to meet you" vbird1
Hello, D.M. Tsai
Nice to meet you in the network.
You are so nice.  byebye!
.    &lt;==这里很重要喔，结束时，最后一行输入小数点 . 即可！
EOT
[root@study ~]#  &lt;==出现提示字符，表示输入完毕了！
```

如此一来，你就已经寄出一封信给 vbird1 这位使用者啰，而且，该信件标题为： nice to meet you，信件内容就如同上面提到的。不过，你或许会觉得 mail 这个程序不好用～ 因为在信件编写的过程中，如果写错字而按下 Enter 进入次行，前一行的数据很难删除ㄟ！ 那怎么办？没关系啦！我们使用数据流重导向啊！呵呵！利用那个小于的符号 （ < ） 就可以达到取代键盘输入的要求了。也就是说，你可以先用 vi 将信件内容编好， 然后再以 mail -s "nice to meet you" vbird1 < filename 来将文件内容传输即可。

例题：请将你的主文件夹下的环境变量文件 （~/.bashrc） 寄给自己！答：mail -s "bashrc file content" dmtsai < ~/.bashrc例题：通过管线命令直接将 ls -al ~ 的内容传给 root 自己！答：ls -al ~ | mail -s "myfile" root

刚刚上面提到的是关于“寄信”的问题，那么如果是要收信呢？呵呵！同样的使用 mail 啊！ 假设我以 vbird1 的身份登陆主机，然后输入 mail 后，会得到什么？

```
[vbird1@study ~]$ mail
Heirloom Mail version 12.5 7/5/10\.  Type ? for help.
"/var/spool/mail/vbird1": 1 message 1 new
&gt;N  1 root                  Wed Jul 22 02:09  20/671   "nice to meet you"
&  &lt;==这里可以输入很多的指令，如果要查阅，输入 ? 即可！
```

在 mail 当中的提示字符是 & 符号喔，别搞错了～输入 mail 之后，我可以看到我有一封信件， 这封信件的前面那个 > 代表目前处理的信件，而在大于符号的右边那个 N 代表该封信件尚未读过， 如果我想要知道这个 mail 内部的指令有哪些，可以在 & 之后输入“？”，就可以看到如下的画面：

```
& ?
               mail commands
type &lt;message list&gt;             type messages
next                            goto and type next message
from &lt;message list&gt;             give head lines of messages
headers                         print out active message headers
delete &lt;message list&gt;           delete messages
undelete &lt;message list&gt;         undelete messages
save &lt;message list&gt; folder      append messages to folder and mark as saved
copy &lt;message list&gt; folder      append messages to folder without marking them
write &lt;message list&gt; file       append message texts to file, save attachments
preserve &lt;message list&gt;         keep incoming messages in mailbox even if saved
Reply &lt;message list&gt;            reply to message senders
reply &lt;message list&gt;            reply to message senders and all recipients
mail addresses                  mail to specific recipients
file folder                     change to another folder
quit                            quit and apply changes to folder
xit                             quit and discard changes made to folder
!                               shell escape
cd &lt;directory&gt;                  chdir to directory or home if none given
list                            list names of all available commands
```

<message list> 指的是每封邮件的左边那个数字啦！而几个比较常见的指令是：

指令	意义
h	列出信件标头；如果要查阅 40 封信件左右的信件标头，可以输入“ h 40 ”
d	删除后续接的信件号码，删除单封是“ d10 ”，删除 20~40 封则为“ d20-40 ”。 不过，这个动作要生效的话，必须要配合 q 这个指令才行（参考下面说明）！
s	将信件储存成文件。例如我要将第 5 封信件的内容存成 ~/mail.file:“s 5 ~/mail.file”
x	或者输入 exit 都可以。这个是“不作任何动作离开 mail 程序”的意思。 不论你刚刚删除了什么信件，或者读过什么，使用 exit 都会直接离开 mail，所以刚刚进行的删除与阅读工作都会无效。 如果您只是查阅一下邮件而已的话，一般来说，建议使用这个离开啦！除非你真的要删除某些信件。
q	相对于 exit 是不动作离开， q 则会实际进行你刚刚所执行的任何动作 （尤其是删除！）

旧版的 CentOS 在使用 mail 读信后，通过 q 离开始，会将已读信件移动到 ~/mbox 中，不过目前 CentOS 7 已经不这么做了！ 所以离开 mail 可以轻松愉快的使用 q 了呢！

13.7 CentOS 7 环境下大量创建帐号的方法

系统上面如果有一堆帐号存在，你怎么判断某些帐号是否存在一些问题？这时需要哪些软件的协助处理比较好？ 另外，如果你跟鸟哥一样，在开学之初或期末之后，经常有需要大量创建帐号、删除帐号的需求时，那么是否要使用 useradd 一行一行指令去创建？ 此外，如果还有需要使用到下一章会介绍到的 quota （磁盘配额） 时，那是否还要额外使用其他机制来创建这些限制值？既然已经学过 shell script 了， 当然写支脚本让它将所有的动作做完最轻松吧！所以啰，下面我们就来聊一聊，如何检查帐号以及创建这个脚本要怎么进行比较好？

13.7.1 一些帐号相关的检查工具

先来检查看看使用者的主文件夹、密码等数据有没有问题？这时会使用到的主要有 pwck 以及 pwconv / pwuconv 等，让我们来了解一下先！

- pwck

pwck 这个指令在检查 /etc/passwd 这个帐号配置文件内的信息，与实际的主文件夹是否存在等信息， 还可以比对 /etc/passwd /etc/shadow 的信息是否一致，另外，如果 /etc/passwd 内的数据字段错误时，会提示使用者修订。 一般来说，我只是利用这个玩意儿来检查我的输入是否正确就是了。

```
[root@study ~]# pwck
user 'ftp': directory '/var/ftp' does not exist
user 'avahi-autoipd': directory '/var/lib/avahi-autoipd' does not exist
user 'pulse': directory '/var/run/pulse' does not exist
pwck: no changes
```

瞧！上面仅是告知我，这些帐号并没有主文件夹，由于那些帐号绝大部分都是系统帐号，确实也不需要主文件夹的，所以，那是“正常的错误！”呵呵！不理他。 ^_^。 相对应的群组检查可以使用 grpck 这个指令的啦！

- pwconv

这个指令主要的目的是在“将 /etc/passwd 内的帐号与密码，移动到 /etc/shadow 当中！” 早期的 Unix 系统当中并没有 /etc/shadow 呢，所以，使用者的登陆密码早期是在 /etc/passwd 的第二栏，后来为了系统安全，才将密码数据移动到 /etc/shadow 内的。使用 pwconv 后，可以：

- 比对 /etc/passwd 及 /etc/shadow ，若 /etc/passwd 内存在的帐号并没有对应的 /etc/shadow 密码时，则 pwconv 会去 /etc/login.defs 取用相关的密码数据，并创建该帐号的 /etc/shadow 数据；
- 若 /etc/passwd 内存在加密后的密码数据时，则 pwconv 会将该密码栏移动到 /etc/shadow 内，并将原本的 /etc/passwd 内相对应的密码栏变成 x ！

一般来说，如果您正常使用 useradd 增加使用者时，使用 pwconv 并不会有任何的动作，因为 /etc/passwd 与 /etc/shadow 并不会有上述两点问题啊！ ^_^。不过，如果手动设置帐号，这个 pwconv 就很重要啰！

- pwunconv

相对于 pwconv ， pwunconv 则是“将 /etc/shadow 内的密码栏数据写回 /etc/passwd 当中，并且删除 /etc/shadow 文件。”这个指令说实在的，最好不要使用啦！ 因为他会将你的 /etc/shadow 删除喔！如果你忘记备份，又不会使用 pwconv 的话，粉严重呢！

- chpasswd

chpasswd 是个挺有趣的指令，他可以“读入未加密前的密码，并且经过加密后， 将加密后的密码写入 /etc/shadow 当中。”这个指令很常被使用在大量创建帐号的情况中喔！ 他可以由 Standard input 读入数据，每笔数据的格式是“ username:password ”。 举例来说，我的系统当中有个使用者帐号为 vbird3 ，我想要更新他的密码 （update） ， 假如他的密码是 abcdefg 的话，那么我可以这样做：

```
[root@study ~]# echo "vbird3:abcdefg" &#124; chpasswd
```

神奇吧！这样就可以更新了呢！在默认的情况中， chpasswd 会去读取 /etc/login.defs 文件内的加密机制，我们 CentOS 7.x 用的是 SHA512， 因此 chpasswd 就默认会使用 SHA512 来加密！如果你想要使用不同的加密机制，那就得要使用 -c 以及 -e 等方式来处理了！ 不过从 CentOS 5.x 开始之后，passwd 已经默认加入了 --stdin 的选项，因此这个 chpasswd 就变得英雄无用武之地了！ 不过，在其他非 Red Hat 衍生的 Linux 版本中，或许还是可以参考这个指令功能来大量创建帐号喔！

13.7.2 大量创建帐号范本（适用 passwd --stdin 选项）

由于 CentOS 7.x 的 passwd 已经提供了 --stdin 的功能，因此如果我们可以提供帐号密码的话， 那么就能够很简单的创建起我们的帐号密码了。下面鸟哥制作一个简单的 script 来执行新增用户的功能喔！

```
[root@study ~]# vim accountadd.sh
#!/bin/bash
# This shell script will create amount of linux login accounts for you.
# 1\. check the "accountadd.txt" file exist? you must create that file manually.
#    one account name one line in the "accountadd.txt" file.
# 2\. use openssl to create users password.
# 3\. User must change his password in his first login.
# 4\. more options check the following url:
# 0410accountmanager.html#manual_amount
# 2015/07/22    VBird
export PATH=/bin:/sbin:/usr/bin:/usr/sbin

# 0\. userinput
usergroup=""                    # if your account need secondary group, add here.
pwmech="openssl"                # "openssl" or "account" is needed.
homeperm="no"                   # if "yes" then I will modify home dir permission to 711

# 1\. check the accountadd.txt file
action="${1}"                   # "create" is useradd and "delete" is userdel.
if [ ! -f accountadd.txt ]; then
    echo "There is no accountadd.txt file, stop here."
        exit 1
fi

[ "${usergroup}" != "" ] && groupadd -r ${usergroup}
rm -f outputpw.txt
usernames=$（cat accountadd.txt）

for username in ${usernames}
do
    case ${action} in
        "create"）
            [ "${usergroup}" != "" ] && usegrp=" -G ${usergroup} " &#124;&#124; usegrp=""
            useradd ${usegrp} ${username}               # 新增帐号
            [ "${pwmech}" == "openssl" ] && usepw=$（openssl rand -base64 6） &#124;&#124;
            echo ${usepw} &#124; passwd --stdin ${username}  # 创建密码
            chage -d 0 ${username}                      # 强制登陆修改密码
            [ "${homeperm}" == "yes" ] && chmod 711 /home/${username}
        echo "username=${username}, password=${usepw}" &gt;&gt; outputpw.txt
            ;;
        "delete"）
            echo "deleting ${username}"
            userdel -r ${username}
            ;;
        *）
            echo "Usage: $0 [create&#124;delete]"
            ;;
    esac
done
```

接下来只要创建 accountadd.txt 这个文件即可！鸟哥创建这个文件里面共有 5 行，你可以自行创建该文件！内容每一行一个帐号。 而是否需要修改密码？是否与帐号相同的信息等等，你可以自由选择！若使用 openssl 自动猜密码时，使用者的密码请由 outputpw.txt 去捞～鸟哥最常作的方法，就是将该文件打印出来，用裁纸机一个帐号一条，交给同学即可！

```
[root@study ~]# vim accountadd.txt
std01
std02
std03
std04
std05

[root@study ~]# sh accountadd.sh create
Changing password for user std01.
passwd: all authentication tokens updated successfully.
....（后面省略）....
```

这支简单的脚本你可以在按如下的链接下载：

- http://linux.vbird.org/linux_basic/0410accountmanager/accountadd.sh

13.8 重点回顾

- Linux 操作系统上面，关于帐号与群组，其实记录的是 UID/GID 的数字而已；
- 使用者的帐号/群组与 UID/GID 的对应，参考 /etc/passwd 及 /etc/group 两个文件
- /etc/passwd 文件结构以冒号隔开，共分为七个字段，分别是“帐号名称、密码、UID、GID、全名、主文件夹、shell”
- UID 只有 0 与非为 0 两种，非为 0 则为一般帐号。一般帐号又分为系统帐号 （1~999）及可登陆者帐号 （大于 1000）
- 帐号的密码已经移动到 /etc/shadow 文件中，该文件权限为仅有 root 可以更动。该文件分为九个字段，内容为“ 帐号名称、加密密码、密码更动日期、密码最小可变动日期、密码最大需变动日期、密码过期前警告日数、密码失效天数、 帐号失效日、保留未使用”
- 使用者可以支持多个群组，其中在新建文件时会影响新文件群组者，为有效群组。而写入 /etc/passwd 的第四个字段者， 称为初始群组。
- 与使用者创建、更改参数、删除有关的指令为：useradd, usermod, userdel等，密码创建则为 passwd；
- 与群组创建、修改、删除有关的指令为：groupadd, groupmod, groupdel 等；
- 群组的观察与有效群组的切换分别为：groups 及 newgrp 指令；
- useradd 指令作用参考的文件有： /etc/default/useradd, /etc/login.defs, /etc/skel/ 等等
- 观察使用者详细的密码参数，可以使用“ chage -l 帐号 ”来处理；
- 使用者自行修改参数的指令有： chsh, chfn 等，观察指令则有： id, finger 等
- ACL 的功能需要文件系统有支持，CentOS 7 默认的 XFS 确实有支持 ACL 功能！
- ACL 可进行单一个人或群组的权限管理，但 ACL 的启动需要有文件系统的支持；
- ACL 的设置可使用 setfacl ，查阅则使用 getfacl ；
- 身份切换可使用 su ，亦可使用 sudo ，但使用 sudo 者，必须先以 visudo 设置可使用的指令；
- PAM 模块可进行某些程序的验证程序！与 PAM 模块有关的配置文件位于 /etc/pam.d/ 及 */etc/security/*
- 系统上面帐号登陆情况的查询，可使用 w, who, last, lastlog 等；
- 线上与使用者交谈可使用 write, wall，离线状态下可使用 mail 传送邮件！

13.9 本章习题

- 情境仿真题一：想将本服务器的帐号分开管理，分为单纯邮件使用，与可登陆系统帐号两种。其中若为纯邮件帐号时， 将该帐号加入 mail 为初始群组，且此帐号不可使用 bash 等 shell 登陆系统。若为可登陆帐号时， 将该帐号加入 youcan 这个次要群组。
    - 目标：了解 /sbin/nologin 的用途；
    - 前提：可自行观察使用者是否已经创建等问题；
    - 需求：需已了解 useradd, groupadd 等指令的用法； 解决方案如下：
    - 预先察看一下两个群组是否存在？

    ```
    [root@study ~]# grep mail /etc/group
    [root@study ~]# grep youcan /etc/group
    [root@study ~]# groupadd youcan
    ```

    可发现 youcan 尚未被创建，因此如上表所示，我们主动去创建这个群组啰。

    - 开始创建三个邮件帐号，此帐号名称为 pop1, pop2, pop3 ，且密码与帐号相同。可使用如下的程序来处理：

    ```
    [root@study ~]# vim popuser.sh
    #!/bin/bash
    for username in pop1 pop2 pop3
    do
        useradd -g mail -s /sbin/nologin -M $username
        echo $username &#124; passwd --stdin $username
    done
    [root@study ~]# sh popuser.sh
    ```

    - 开始创建一般帐号，只是这些一般帐号必须要能够登陆，并且需要使用次要群组的支持！所以：

    ```
    [root@study ~]# vim loginuser.sh
    #!/bin/bash
    for username in youlog1 youlog2 youlog3
    do
        useradd -G youcan -s /bin/bash -m $username
        echo $username &#124; passwd --stdin $username
    done
    [root@study ~]# sh loginuser.sh
    ```

    - 这样就将帐号分开管理了！非常简单吧！

简答题部分

- root 的 UID 与 GID 是多少？而基于这个理由，我要让 test 这个帐号具有 root 的权限，

应该怎么作？root 的 UID 与 GID 均为 0 ，所以要让 test 变成 root 的权限，那么就将 /etc/passwd 里面， test 的 UID 与 GID 字段变成 0 即可！

- 假设我是一个系统管理员，我有一个用户最近不乖，所以我想暂时将他的帐号停掉， 让他近期无法进行任何动作，等到未来他乖一点之后，我再将他的帐号启用，请问：我可以怎么作比较好？？由于这个帐号是暂时失效的，所以不能使用 userdel 来删除，否则很麻烦！那么应该如何设置呢？再回去瞧一瞧 /etc/shadow 的架构，可以知道有这几个可使用的方法：
    - 将 /etc/passwd 的 shell 字段写成 /sbin/nologin ，即可让该帐号暂时无法登陆主机；
    - 将 /etc/shadow 内的密码字段，增加一个 * 号在最前面，这样该帐号亦无法登陆！
    - 将 /etc/shadow 的第八个字段关于帐号取消日期的那个，设置小于目前日期的数字，那么他就无法登陆系统了！
- 我在使用 useradd 的时候，新增的帐号里面的 UID, GID 还有其他相关的密码控制，都是在哪几个文件里面设置的？在 /etc/login.defs 还有 /etc/default/useradd 里面规定好的！
- 我希望我在设置每个帐号的时候（ 使用 useradd ），默认情况中，他们的主文件夹就含有一个名称为 www 的子目录，我应该怎么作比较好？由于使用 useradd 的时候，会自动以 /etc/skel 做为默认的主文件夹，所以，我可以在 /etc/skel 里面新增加一个名称为 www 的目录即可！
- 简单说明系统帐号与一般使用者帐号的差别？一般而言，为了让系统能够顺利以较小的权限运行，系统会有很多帐号， 例如 mail, bin, adm 等等。而为了确保这些帐号能够在系统上面具有独一无二的权限， 一般来说 Linux 都会保留一些 UID 给系统使用。在 CentOS 5.x 上面，小于 500 以下的帐号 （UID） 即是所谓的 System account。
- 简单说明，为何 CentOS 创建使用者时，他会主动的帮使用者创建一个群组，而不是使用 /etc/default/useradd 的设置？不同的 linux distributions 对于使用者 group 的创建机制并不相同。主要的机制分为：
    - Public group schemes: 使用者将会直接给予一个系统指定的群组，一般来说即是 users ， 可以 SuSE Server 9 为代表；
    - Private group schemes: 系统会创建一个与帐号一样的群组名称！以 CentOS 7.x 为例！
- 如何创建一个使用者名称 alex, 他所属群组为 alexgroup, 预计使用 csh, 他的全名为 "Alex Tsai"， 且他还得要加入 users 群组当中！groupadd alexgroup useradd -c "Alex Tsai" -g alexgroup -G users -m alex 务必先创建群组，才能够创建使用者喔！
- 由于种种因素，导致你的使用者主文件夹以后都需要被放置到 /account 这个目录下。 请问，我该如何作，可以让使用 useradd 时，默认的主文件夹就指向 /account ？最简单的方法，编辑 /etc/default/useradd ，将里头的 HOME=/home 改成 HOME=/account 即可。
- 我想要让 dmtsai 这个使用者，加入 vbird1, vbird2, vbird3 这三个群组，且不影响 dmtsai 原本已经支持的次要群组时，该如何动作？usermod -a -G vbird1,vbird2,vbird3 dmtsai

13.10 参考资料与延伸阅读

- [1]最完整与详细的密码档说明，可参考各 distribution 内部的 man page。 本文中以 CentOS 7.x 的“ man 5 passwd ”及“ man 5 shadow ”的内容说明；
- [2]MD5, DES, SHA 均为加密的机制，详细的解释可参考维基百科的说明：
  - MD5：http://zh.wikipedia.org/wiki/MD5
  - DES：http://en.wikipedia.org/wiki/Data_Encryption_Standard
  - SHA家族：https://en.wikipedia.org/wiki/Secure_Hash_Algorithm在早期的 Linux 版本中，主要使用 MD5 加密演算法，近期则使用 SHA512 作为默认演算法。
- [3]telnet 与 ssh 都是可以由远端用户主机连线到 Linux 服务器的一种机制！详细数据可查询鸟站文章： 远端连线服务器：http://linux.vbird.org/linux_server/0310telnetssh.php
- [4]详细的说明请参考 man sudo ，然后以 5 作为关键字搜寻看看即可了解。
- [5]详细的 PAM 说明可以参考如下链接： 维基百科：http://en.wikipedia.org/wiki/Pluggable_Authentication_Modules Linux-PAM网页：http://www.kernel.org/pub/linux/libs/pam/

2002/05/15：第一次完成 2003/02/10：重新编排与加入 FAQ 2005/08/25：加入一个大量创建帐号的实例，简单说明一下而已！ 2005/08/29：将原本的旧文放置到 此处 2005/08/31：因为 userconf 已经不再这么好用了，使用指令模式比较简单，所以，将他拿掉了～ 2005/09/05：终于将大量创建帐号的那支程序写完了～真是高兴啊！ 2006/03/02：更新使用者 UID 号码，由 65535 升级到 2^32-1 这么大！ 2007/04/15：原本写的 /etc/pam.d/limits.conf 错了！应该是 /etc/security/limits.conf 才对！ 2008/04/28：sudo 关于密码重新输入的部分写错了！已经更新，在这里查阅看看。感谢网友 superpmo 的告知！ 2009/02/18：将基于 FC4 版本的旧文章移动到 此处。 2009/02/26：加入 chage 以及“ chage -d 0 帐号”的功能！ 2009/02/27：加入 acl 的控制项目！ 2009/03/04：加入一个简单的帐号新增范例，以及修改原本的帐号新增范例！ 2009/04/28：取消 sudo 内的 -c 选项功能说明！之前说的是错的～ 2009/09/09：加入一些仿真题，修改一些语词的用法。 2010/04/27：情境仿真第三步骤的程序脚本错了！原本是：“useradd -G youcan -s -m $username”！感谢 linux_task 兄的说明呢！ 2015/07/17：将旧的基于 CentOS 5 的版本备份到这里。 2015/07/27：忘记加入 authconfig-tui 的说明！今日补上！

第十四章、磁盘配额（Quota）与进阶文件系统管理

最近更新日期：20//

如果您的 Linux 服务器有多个用户经常存取数据时，为了维护所有使用者在硬盘容量的公平使用，磁盘配额 （Quota） 就是一项非常有用的工具！另外，如果你的用户常常抱怨磁盘容量不够用，那么更进阶的文件系统就得要学习学习。 本章我们会介绍磁盘阵列 （RAID） 及逻辑卷轴文件系统 （LVM），这些工具都可以帮助你管理与维护使用者可用的磁盘容量喔！

14.1 磁盘配额 （Quota） 的应用与实作

Quota 这个玩意儿就字面上的意思来看，就是有多少“限额”的意思啦！如果是用在零用钱上面， 就是类似“有多少零用钱一个月”的意思之类的。如果是在计算机主机的磁盘使用量上呢？以 Linux 来说，就是有多少容量限制的意思啰。我们可以使用 quota 来让磁盘的容量使用较为公平， 下面我们会介绍什么是 quota ，然后以一个完整的范例来介绍 quota 的实作喔！

14.1.1 什么是 Quota

在 Linux 系统中，由于是多用户多任务的环境，所以会有多人共同使用一个硬盘空间的情况发生， 如果其中有少数几个使用者大量的占掉了硬盘空间的话，那势必压缩其他使用者的使用权力！ 因此管理员应该适当的限制硬盘的容量给使用者，以妥善的分配系统资源！避免有人抗议呀！

举例来说，我们使用者的默认主文件夹都是在 /home 下面，如果 /home 是个独立的 partition ， 假设这个分区有 10G 好了，而 /home 下面共有 30 个帐号，也就是说，每个使用者平均应该会有 333MB 的空间才对。 偏偏有个使用者在他的主文件夹下面塞了好多只影片，占掉了 8GB 的空间，想想看，是否造成其他正常使用者的不便呢？ 如果想要让磁盘的容量公平的分配，这个时候就得要靠 quota 的帮忙啰！

- Quota 的一般用途 [1]

quota 比较常使用的几个情况是：

- 针对 WWW server ，例如：每个人的网页空间的容量限制！
- 针对 mail server，例如：每个人的邮件空间限制。
- 针对 file server，例如：每个人最大的可用网络硬盘空间 （教学环境中最常见！）

上头讲的是针对网络服务的设计，如果是针对 Linux 系统主机上面的设置那么使用的方向有下面这一些：

- 限制某一群组所能使用的最大磁盘配额 （使用群组限制）： 你可以将你的主机上的使用者分门别类，有点像是目前很流行的付费与免付费会员制的情况， 你比较喜好的那一群的使用配额就可以给高一些！呵呵！ ^_^...

- 限制某一使用者的最大磁盘配额 （使用使用者限制）： 在限制了群组之后，你也可以再继续针对个人来进行限制，使得同一群组之下还可以有更公平的分配！

- 限制某一目录 （directory, project） 的最大磁盘配额： 在旧版的 CentOS 当中，使用的默认文件系统为 EXT 家族，这种文件系统的磁盘配额主要是针对整个文件系统来处理，所以大多针对“挂载点”进行设计。 新的 xfs 可以使用 project 这种模式，就能够针对个别的目录 （非文件系统喔） 来设计磁盘配额耶！超棒的！

大概有这些实际的用途啦！基本上，quota 就是在回报管理员磁盘使用率以及让管理员管理磁盘使用情况的一个工具就是了！ 比较特别的是，XFS 的 quota 是整合到文件系统内，并不是其他外挂的程序来管理的，因此，通过 quota 来直接回报磁盘使用率，要比 unix 工具来的快速！ 举例来说， du 这东西会重新计算目录下的磁盘使用率，但 xfs 可以通过 xfs_quota 来直接回报各目录使用率，速度上是快非常多！

- Quota 的使用限制

虽然 quota 很好用，但是使用上还是有些限制要先了解的：

- 在 EXT 文件系统家族仅能针对整个 filesystem： EXT 文件系统家族在进行 quota 限制的时候，它仅能针对整个文件系统来进行设计，无法针对某个单一的目录来设计它的磁盘配额。 因此，如果你想要使用不同的文件系统进行 quota 时，请先搞清楚该文件系统支持的情况喔！因为 XFS 已经可以使用 project 模式来设计不同目录的磁盘配额。

- 核心必须支持 quota ： Linux 核心必须有支持 quota 这个功能才行：如果你是使用 CentOS 7.x 的默认核心， 嘿嘿！那恭喜你了，你的系统已经默认有支持 quota 这个功能啰！如果你是自行编译核心的， 那么请特别留意你是否已经“真的”打开了 quota 这个功能？否则下面的功夫将全部都视为“白工”。

- 只对一般身份使用者有效： 这就有趣了！并不是所有在 Linux 上面的帐号都可以设置 quota 呢，例如 root 就不能设置 quota ， 因为整个系统所有的数据几乎都是他的啊！ ^_^

- 若启用 SELinux，非所有目录均可设置 quota ： 新版的 CentOS 默认都有启用 SELinux 这个核心功能，该功能会加强某些细部的权限控制！由于担心管理员不小心设置错误，因此默认的情况下， quota 似乎仅能针对 /home 进行设置而已～因此，如果你要针对其他不同的目录进行设置，请参考到后续章节查阅解开 SELinux 限制的方法喔！ 这就不是 quota 的问题了...

新版的 CentOS 使用的 xfs 确实比较有趣！不但无须额外的 quota 纪录档，也能够针对文件系统内的不同目录进行配置！相当有趣！ 只是不同的文件系统在 quota 的处理情况上不太相同，因此这里要特别强调，进行 quota 前，先确认你的文件系统吧！

- Quota 的规范设置项目：

quota 这玩意儿针对 XFS filesystem 的限制项目主要分为下面几个部分：

- 分别针对使用者、群组或个别目录 （user, group & project）：

XFS 文件系统的 quota 限制中，主要是针对群组、个人或单独的目录进行磁盘使用率的限制！

- 容量限制或文件数量限制 （block 或 inode）：

我们在第七章谈到文件系统中，说到文件系统主要规划为存放属性的 inode 与实际文件数据的 block 区块，Quota 既然是管理文件系统，所以当然也可以管理 inode 或 block 啰！ 这两个管理的功能为：

- 限制 inode 用量：可以管理使用者可以创建的“文件数量”；
- 限制 block 用量：管理使用者磁盘容量的限制，较常见为这种方式。

- 柔性劝导与硬性规定 （soft/hard）：

既然是规范，当然就有限制值。不管是 inode/block ，限制值都有两个，分别是 soft 与 hard。 通常 hard 限制值要比 soft 还要高。举例来说，若限制项目为 block ，可以限制 hard 为 500MBytes 而 soft 为 400MBytes。这两个限值的意义为：

- hard：表示使用者的用量绝对不会超过这个限制值，以上面的设置为例， 使用者所能使用的磁盘容量绝对不会超过 500MBytes ，若超过这个值则系统会锁住该用户的磁盘使用权；
- soft：表示使用者在低于 soft 限值时 （此例中为 400MBytes），可以正常使用磁盘，但若超过 soft 且低于 hard 的限值 （介于 400~500MBytes 之间时），每次使用者登陆系统时，系统会主动发出磁盘即将爆满的警告讯息， 且会给予一个宽限时间 （grace time）。不过，若使用者在宽限时间倒数期间就将容量再次降低于 soft 限值之下， 则宽限时间会停止。

- 会倒数计时的宽限时间 （grace time）：

刚刚上面就谈到宽限时间了！这个宽限时间只有在使用者的磁盘用量介于 soft 到 hard 之间时，才会出现且会倒数的一个咚咚！ 由于达到 hard 限值时，使用者的磁盘使用权可能会被锁住。为了担心使用者没有注意到这个磁盘配额的问题， 因此设计了 soft 。当你的磁盘用量即将到达 hard 且超过 soft 时，系统会给予警告，但也会给一段时间让使用者自行管理磁盘。 一般默认的宽限时间为七天，如果七天内你都不进行任何磁盘管理，那么 soft 限制值会即刻取代 hard 限值来作为 quota 的限制。

以上面设置的例子来说，假设你的容量高达 450MBytes 了，那七天的宽限时间就会开始倒数， 若七天内你都不进行任何删除文件的动作来替你的磁盘用量瘦身， 那么七天后你的磁盘最大用量将变成 400MBytes （那个 soft 的限制值），此时你的磁盘使用权就会被锁住而无法新增文件了。

整个 soft, hard, grace time 的相关性我们可以用下面的图示来说明：

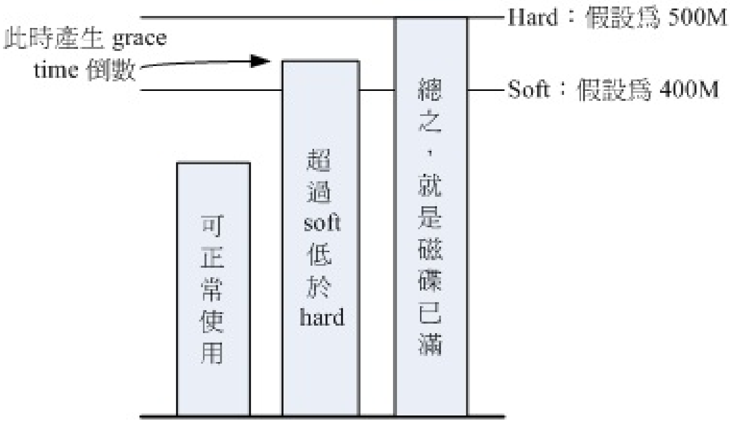


图14.1.1、soft, hard, grace time 的相关性

图中的长条图为使用者的磁盘容量，soft/hard 分别是限制值。只要小于 400M 就一切 OK ，若高于 soft 就出现 grace time 并倒数且等待使用者自行处理，若到达 hard 的限制值， 那我们就搬张小板凳等着看好戏啦！嘿嘿！^_^！这样图示有清楚一点了吗？

14.1.2 一个 XFS 文件系统的 Quota 实作范例

坐而言不如起而行啊，所以这里我们使用一个范例来设计一下如何处理 Quota 的设置流程。

- 目的与帐号：现在我想要让我的专题生五个为一组，这五个人的帐号分别是 myquota1, myquota2, myquota3, myquota4, myquota5，这五个用户的密码都是 password ，且这五个用户所属的初始群组都是 myquotagrp 。 其他的帐号属性则使用默认值。
- 帐号的磁盘容量限制值：我想让这五个用户都能够取得 300MBytes 的磁盘使用量（hard），文件数量则不予限制。 此外，只要容量使用率超过 250MBytes ，就予以警告（soft）。
- 群组的限额 （option 1）：由于我的系统里面还有其他用户存在，因此我仅承认 myquotagrp 这个群组最多仅能使用 1GBytes 的容量。 这也就是说，如果 myquota1, myquota2, myquota3 都用了 280MBytes 的容量了，那么其他两人最多只能使用（1000MB - 280x3 = 160MB） 的磁盘容量啰！这就是使用者与群组同时设置时会产生的后果。
- 共享目录限额 （option 2）：另一种设置方式，每个用户还是具有自己独立的容量限止，但是这五个人的专题共享目录在 /home/myquota 这里，该目录请设置为其他人没有任何权限的共享目录空间，仅有 myquotagrp 群组拥有全部的权限。 且无论如何，该目录最多仅能够接受 500MBytes 的容量。请注意，群组 （group） 的限制与目录（directory/project） 无法同时并存喔！ 所以下面的流程中，我们会先以群组来设计，然后再以目录限制来进一步说明！
- 宽限时间的限制：最后，我希望每个使用者在超过 soft 限制值之后，都还能够有 14 天的宽限时间。

好了，那你怎么规范帐号以及相关的 Quota 设置呢？首先，在这个小节我们先来将帐号相关的属性、参数及其他环境搞定再说吧！

```
# 制作帐号环境时，由于有五个帐号，因此鸟哥使用 script 来创建环境！
[root@study ~]# vim addaccount.sh
#!/bin/bash
# 使用 script 来创建实验 quota 所需的环境
groupadd myquotagrp
for username in myquota1 myquota2 myquota3 myquota4 myquota5
do
    useradd -g myquotagrp $username
    echo "password" &#124; passwd --stdin $username
done
mkdir /home/myquota
chgrp myquotagrp /home/myquota
chmod 2770 /home/myquota

[root@study ~]# sh addaccount.sh
```

接下来，就让我们来实作 Quota 的练习吧！

14.1.3 实作 Quota 流程-1：文件系统的支持与观察

前面我们就谈到，要使用 Quota 必须要核心与文件系统支持才行！假设你已经使用了默认支持 Quota 的核心， 那么接下来就是要启动文件系统的支持啦！但是要注意，我们这边是以 XFS 文件系统为例的，如果你使用的是 EXT 家族，请找前一版的书籍说明喔！ 此外，不要在根目录下面进行 quota 设计喔！因为文件系统会变得太复杂！因此，下面我们是以 /home 这个 xfs 文件系统为例的！ 当然啦，首先就是要来检查看看！

```
[root@study ~]# df -hT /home
Filesystem              Type  Size  Used Avail Use% Mounted on
/dev/mapper/centos-home xfs   5.0G   67M  5.0G   2% /home
```

从上面的数据来看，鸟哥这部主机的 /home 确实是独立的 filesystem，而且确实是使用了 xfs 文件系统！所以可以使用下面的流程啰！ 此外，由于 VFAT 文件系统并不支持 Linux Quota 功能，所以我们得要使用 mount 查询一下 /home 的文件系统为何才行啊！

在过去的版本中，管理员似乎可以通过 mount -o remount 的机制来重新挂载启动 quota 的功能，不过 XFS 文件系统的 quota 似乎是在挂载之初就宣告了， 因此无法使用 remount 来重新启动 quota 功能，一定得要写入 /etc/fstab 当中，或者是在初始挂载过程中加入这个项目，否则不会生效喔！那就来瞧瞧鸟哥改了 fstab 成为怎样吧！

```
[root@study ~]# vim /etc/fstab
/dev/mapper/centos-home  /home  xfs  defaults,usrquota,grpquota   0 0
# 其他项目鸟哥并没有列出来！重点在于第四字段！于 default 后面加上两个参数！

[root@study ~]# umount /home
[root@study ~]# mount -a
[root@study ~]# mount &#124; grep home
/dev/mapper/centos-home on /home type xfs （rw,relatime,seclabel,attr2,inode64,usrquota,g
```

基本上，针对 quota 限制的项目主要有三项，如下所示：

- uquota/usrquota/quota：针对使用者帐号的设置
- gquota/grpquota：针对群组的设置
- pquota/prjquota：针对单一目录的设置，但是不可与 grpquota 同时存在！

还是要再次的强调，修改完 /etc/fstab 后，务必要测试一下！若有发生错误得要赶紧处理！ 因为这个文件如果修改错误，是会造成无法开机完全的情况啊！切记切记！最好使用 vim 来修改啦！ 因为会有语法的检验，就不会让你写错字了！此外，由于一般用户的主文件夹在 /home 里面，因此针对这个项目的卸载时， 一定要将所有一般帐号的身份登出，否则肯定无法卸载喔！留意留意！

14.1.4 实作 Quota 流程-2：观察 Quota 报告数据

制作文件系统支持之后，当然得要来瞧一瞧到底有没有正确的将 quota 的管理数据列出来才好！这时我们得要使用 xfs_quota 这个指令才行！这个指令真的是挺复杂的，因为全部的 quota 实作都是这个指令耶！所以里面的参数有够多！ 不过稍微观察一下即可！先让我们来谈谈观察目前 quota 的报告内容吧！

```
[root@study ~]# xfs_quota -x -c "指令" [挂载点]
选项与参数：
-x  ：专家模式，后续才能够加入 -c 的指令参数喔！
-c  ：后面加的就是指令，这个小节我们先来谈谈数据回报的指令
指令：
      print ：单纯的列出目前主机内的文件系统参数等数据
      df    ：与原本的 df 一样的功能，可以加上 -b (block) -i (inode) -h (加上单位) 等
      report：列出目前的 quota 项目，有 -ugr (user/group/project) 及 -bi 等数据
      state ：说明目前支持 quota 的文件系统的信息，有没有起动相关项目等

范例一：列出目前系统的各的文件系统，以及文件系统的 quota 挂载参数支持
[root@study ~]# xfs_quota -x -c "print"
Filesystem          Pathname
/                   /dev/mapper/centos-root
/srv/myproject      /dev/vda4
/boot               /dev/vda2
/home               /dev/mapper/centos-home (uquota, gquota)  # 所以这里就有显示支持啰

范例二：列出目前 /home 这个支持 quota 的载点文件系统使用情况
[root@study ~]# xfs_quota -x -c "df -h" /home
Filesystem     Size   Used  Avail Use% Pathname
/dev/mapper/centos-home
               5.0G  67.0M   4.9G   1% /home
# 如上所示，其实跟原本的 df 差不多啦！只是会更正确就是了。

范例三：列出目前 /home 的所有用户的 quota 限制值
[root@study ~]# xfs_quota -x -c "report -ubih" /home
User quota on /home (/dev/mapper/centos-home)
                        Blocks                            Inodes
User ID      Used   Soft   Hard Warn/Grace     Used   Soft   Hard Warn/Grace
---------- --------------------------------- ---------------------------------
root           4K      0      0  00 [------]      4      0      0  00 [------]
dmtsai      34.0M      0      0  00 [------]    432      0      0  00 [------]
.....(中间省略).....
myquota1      12K      0      0  00 [------]      7      0      0  00 [------]
myquota2      12K      0      0  00 [------]      7      0      0  00 [------]
myquota3      12K      0      0  00 [------]      7      0      0  00 [------]
myquota4      12K      0      0  00 [------]      7      0      0  00 [------]
myquota5      12K      0      0  00 [------]      7      0      0  00 [------]
# 所以列出了所有用户的目前的文件使用情况，并且列出设置值。注意，最上面的 Block
# 代表这个是 block 容量限制，而 inode 则是文件数量限制喔。另外，soft/hard 若为 0，代表没限制

范例四：列出目前支持的 quota 文件系统是否有起动了 quota 功能？
[root@study ~]# xfs_quota -x -c "state"
User quota state on /home (/dev/mapper/centos-home)
  Accounting: ON    # 有启用计算功能
  Enforcement: ON   # 有实际 quota 管制的功能
  Inode: #1568 (4 blocks, 4 extents)  # 上面四行说明的是有启动 user 的限制能力
Group quota state on /home (/dev/mapper/centos-home)
  Accounting: ON
  Enforcement: ON
  Inode: #1569 (5 blocks, 5 extents)  # 上面四行说明的是有启动 group 的限制能力
Project quota state on /home (/dev/mapper/centos-home)
  Accounting: OFF
  Enforcement: OFF
  Inode: #1569 (5 blocks, 5 extents)  # 上面四行说明的是 project 并未支持
Blocks grace time: [7 days 00:00:30]  # 下面则是 grace time 的项目
Inodes grace time: [7 days 00:00:30]
Realtime Blocks grace time: [7 days 00:00:30]
```

在默认的情况下， xfs_quota 的 report 指令会将支持的 user/group/prject 相关数据列出来，如果只是想要某个特定的项目， 例如我们上面要求仅列出用户的数据时，就在 report 后面加上 -u 即可喔！这样就能够观察目前的相关设置信息了。 要注意，限制的项目有 block/inode 同时可以针对每个项目来设置 soft/hard 喔！接下来实际的设置看看吧！

14.1.5 实作 Quota 流程-3：限制值设置方式

确认文件系统的 quota 支持顺利启用后，也能够观察到相关的 quota 限制，接下来就是要实际的给予用户/群组限制啰！ 回去瞧瞧，我们需要每个用户 250M/300M 的容量限制，群组共 950M/1G 的容量限制，同时 grace time 设置为 14 天喔！ 实际的语法与设置流程来瞧瞧：

```
[root@study ~]# xfs_quota -x -c "limit [-ug] b[soft&#124;hard]=N i[soft&#124;hard]=N name
[root@study ~]# xfs_quota -x -c "timer [-ug] [-bir] Ndays"
选项与参数：
limit ：实际限制的项目，可以针对 user/group 来限制，限制的项目有
        bsoft/bhard : block 的 soft/hard 限制值，可以加单位
        isoft/ihard : inode 的 soft/hard 限制值
        name        : 就是用户/群组的名称啊！
timer ：用来设置 grace time 的项目喔，也是可以针对 user/group 以及 block/inode 设置

范例一：设置好用户们的 block 限制值 （题目中没有要限制 inode 啦！）
[root@study ~]# xfs_quota -x -c "limit -u bsoft=250M bhard=300M myquota1" /home
[root@study ~]# xfs_quota -x -c "limit -u bsoft=250M bhard=300M myquota2" /home
[root@study ~]# xfs_quota -x -c "limit -u bsoft=250M bhard=300M myquota3" /home
[root@study ~]# xfs_quota -x -c "limit -u bsoft=250M bhard=300M myquota4" /home
[root@study ~]# xfs_quota -x -c "limit -u bsoft=250M bhard=300M myquota5" /home
[root@study ~]# xfs_quota -x -c "report -ubih" /home
User quota on /home （/dev/mapper/centos-home）
                        Blocks                            Inodes
User ID      Used   Soft   Hard Warn/Grace     Used   Soft   Hard Warn/Grace
---------- --------------------------------- ---------------------------------
myquota1      12K   250M   300M  00 [------]      7      0      0  00 [------]

范例二：设置好 myquotagrp 的 block 限制值
[root@study ~]# xfs_quota -x -c "limit -g bsoft=950M bhard=1G myquotagrp" /home
[root@study ~]# xfs_quota -x -c "report -gbih" /home
Group quota on /home （/dev/mapper/centos-home）
                        Blocks                            Inodes
Group ID     Used   Soft   Hard Warn/Grace     Used   Soft   Hard Warn/Grace
---------- --------------------------------- ---------------------------------
myquotagrp    60K   950M     1G  00 [------]     36      0      0  00 [------]

范例三：设置一下 grace time 变成 14 天吧！
[root@study ~]# xfs_quota -x -c "timer -ug -b 14days" /home
[root@study ~]# xfs_quota -x -c "state" /home
User quota state on /home （/dev/mapper/centos-home）
.....（中间省略）.....
Blocks grace time: [14 days 00:00:30]
Inodes grace time: [7 days 00:00:30]
Realtime Blocks grace time: [7 days 00:00:30]

范例四：以 myquota1 用户测试 quota 是否真的实际运行呢？
[root@study ~]# su - myquota1
[myquota1@study ~]$ dd if=/dev/zero of=123.img bs=1M count=310
dd: error writing ‘123.img’: Disk quota exceeded
300+0 records in
299+0 records out
314552320 Bytes （315 MB） copied, 0.181088 s, 1.7 GB/s
[myquota1@study ~]$ ll -h
-rw-r--r--. 1 myquota1 myquotagrp 300M Jul 24 21:38 123.img

[myquota1@study ~]$ exit
[root@study ~]# xfs_quota -x -c "report -ubh" /home
User quota on /home （/dev/mapper/centos-home）
                        Blocks
User ID      Used   Soft   Hard Warn/Grace
---------- ---------------------------------
myquota1     300M   250M   300M  00 [13 days]
myquota2      12K   250M   300M  00 [------]
# 因为 myquota1 的磁盘用量已经破表，所以当然就会出现那个可怕的 grace time 啰！
```

这样就直接制做好 quota 啰！看起来也是挺简单啦！

14.1.6 实作 Quota 流程-4：project 的限制 （针对目录限制）（Optional）

现在让我们来想一想，如果需要限制的是目录而不是群组时，那该如何处理呢？举例来说，我们要限制的是 /home/myquota 这个目录本身， 而不是针对 myquotagrp 这个群组啊！这两种设置方法的意义不同喔！例如，前一个小节谈到的测试范例来说， myquota1 已经消耗了 300M 的容量，而 /home/myquota 其实还没有任何的使用量 （因为在 myquota1 的主文件夹做的 dd 指令）。 不过如果你使用了 xfs_quota -x -c "report -h" /home 这个指令来查看，就会发现其实 myquotagrp 已经用掉了 300M 了！ 如此一来，对于目录的限制来说，就不会有效果！

为了解决这个问题，因此我们这个小节要来设置那个很有趣的 project 项目！只是这个项目不可以跟 group 同时设置喔！ 因此我们得要取消 group 设置并且加入 project 设置才行。那就来实验看看。

- 修改 /etc/fstab 内的文件系统支持参数

首先，要将 grpquota 的参数取消，然后加入 prjquota ，并且卸载 /home 再重新挂载才行！那就来测试看看！

```
# 1\. 先修改 /etc/fstab 的参数，并启动文件系统的支持
[root@study ~]# vim /etc/fstab
/dev/mapper/centos-home /home xfs  defaults,usrquota,grpquota,prjquota  0 0
# 记得， grpquota 与 prjquota 不可同时设置喔！所以上面删除 grpquota 加入 prjquota

[root@study ~]# umount /home
[root@study ~]# mount -a
[root@study ~]# xfs_quota -x -c "state"
User quota state on /home （/dev/mapper/centos-home）
  Accounting: ON
  Enforcement: ON
  Inode: #1568 （4 blocks, 4 extents）
Group quota state on /home （/dev/mapper/centos-home）
  Accounting: OFF         &lt;==已经取消啰！
  Enforcement: OFF
  Inode: N/A
Project quota state on /home （/dev/mapper/centos-home）
  Accounting: ON          &lt;==确实启动啰！
  Enforcement: ON
  Inode: N/A
Blocks grace time: [7 days 00:00:30]
Inodes grace time: [7 days 00:00:30]
Realtime Blocks grace time: [7 days 00:00:30]
```

- 规范目录、专案名称（project）与专案 ID

目录的设置比较奇怪，他必须要指定一个所谓的“专案名称、专案识别码”来规范才行！而且还需要用到两个配置文件！ 这个让鸟哥觉得比较怪一些就是了。现在，我们要规范的目录是 /home/myquota 目录，这个目录我们给个 myquotaproject 的专案名称， 这个专案名称给个 11 的识别码，这个都是自己指定的，若不喜欢就自己指定另一个吧！鸟哥的指定方式如下：

```
# 2.1 指定专案识别码与目录的对应在 /etc/projects
[root@study ~]# echo "11:/home/myquota" &gt;&gt; /etc/projects

# 2.2 规范专案名称与识别码的对应在 /etc/projid
[root@study ~]# echo "myquotaproject:11" &gt;&gt; /etc/projid

# 2.3 初始化专案名称
[root@study ~]# xfs_quota -x -c "project -s myquotaproject"
Setting up project myquotaproject （path /home/myquota）...
Processed 1 （/etc/projects and cmdline） paths for project myquotaproject with recursion
depth infinite （-1）.    # 会闪过这些讯息！是 OK 的！别担心！

[root@study ~]# xfs_quota -x -c "print " /home
Filesystem          Pathname
/home               /dev/mapper/centos-home （uquota, pquota）
/home/myquota       /dev/mapper/centos-home （project 11, myquotaproject）
# 这个 print 功能很不错！可以完整的查看到相对应的各项文件系统与 project 目录对应！

[root@study ~]# xfs_quota -x -c "report -pbih " /home
Project quota on /home （/dev/mapper/centos-home）
                        Blocks                            Inodes
Project ID       Used   Soft   Hard Warn/Grace     Used   Soft   Hard Warn/Grace
---------- --------------------------------- ---------------------------------
myquotaproject      0      0      0  00 [------]      1      0      0  00 [------]
# 喔耶！确定有抓到这个专案名称啰！接下来准备设置吧！
```

- 实际设置规范与测试

依据本章的说明，我们要将 /home/myquota 指定为 500M 的容量限制，那假设到 450M 为 soft 的限制好了！ 那么设置就会变成这样啰：

```
# 3.1 先来设置好这个 project 吧！设置的方式同样使用 limit 的 bsoft/bhard 喔！：
[root@study ~]# xfs_quota -x -c "limit -p bsoft=450M bhard=500M myquotaproject" /home
[root@study ~]# xfs_quota -x -c "report -pbih " /home
Project quota on /home （/dev/mapper/centos-home）
                            Blocks                            Inodes
Project ID       Used   Soft   Hard Warn/Grace     Used   Soft   Hard Warn/Grace
---------- --------------------------------- ---------------------------------
myquotaproject      0   450M   500M  00 [------]      1      0      0  00 [------]

[root@study ~]# dd if=/dev/zero of=/home/myquota/123.img bs=1M count=510
dd: error writing '/home/myquota/123.img': No space left on device
501+0 records in
500+0 records out
524288000 Bytes （524 MB） copied, 0.96296 s, 544 MB/s
# 你看！连 root 在该目录下面创建文件时，也会被挡掉耶！这才是完整的针对目录的规范嘛！赞！
```

这样就设置好了啰！未来如果你还想要针对某些个目录进行限制，那么就修改 /etc/projects, /etc/projid 设置一下规范， 然后直接处理目录的初始化与设置，就完成设置了！好简单！

当鸟哥跟同事分享这个 project 的功能时，强者我同事蔡董大大说，刚刚好！他有些朋友要求在 WWW 的服务中，要针对某些目录进行容量的限制！ 但是因为容量之前仅针对用户进行限制，如此一来，由于 WWW 服务都是一个名为 httpd 的帐号管理的，因此所有 WWW 服务所产生的文件数据， 就全部属于 httpd 这个帐号，那就无法针对某些特定的目录进行限制了。 有了这个 project 之后，就能够针对不同的目录做容量限制！ 而不用管在里头创建文件的文件拥有者！哇！这眞是太棒了！实务应用给各位了解啰！ ^_^

14.1.7 XFS quota 的管理与额外指令对照表

不管多完美的系统，总是需要可能的突发状况应付手段啊！所以，接下来我们就来谈谈，那么万一如果你需要暂停 quota 的限制， 或者是重新启动 quota 的限制时，该如何处理呢？还是使用 xfs_quota 啦！增加几个内部指令即可：

- disable：暂时取消 quota 的限制，但其实系统还是在计算 quota 中，只是没有管制而已！应该算最有用的功能啰！
- enable：就是回复到正常管制的状态中，与 disable 可以互相取消、启用！
- off：完全关闭 quota 的限制，使用了这个状态后，你只有卸载再重新挂载才能够再次的启动 quota 喔！也就是说， 用了 off 状态后，你无法使用 enable 再次复原 quota 的管制喔！注意不要乱用这个状态！一般建议用 disable 即可，除非你需要执行 remove 的动作！
- remove：必须要在 off 的状态下才能够执行的指令～这个 remove 可以“移除”quota 的限制设置，例如要取消 project 的设置， 无须重新设置为 0 喔！只要 remove -p 就可以了！

现在就让我们来测试一下管理的方式吧：

```
# 1\. 暂时关闭 XFS 文件系统的 quota 限制功能
[root@study ~]# xfs_quota -x -c "disable -up" /home
[root@study ~]# xfs_quota -x -c "state" /home
User quota state on /home （/dev/mapper/centos-home）
 Accounting: ON
  Enforcement: OFF   &lt;== 意思就是有在计算，但没有强制管制的意思
  Inode: #1568 （4 blocks, 4 extents）
Group quota state on /home （/dev/mapper/centos-home）
  Accounting: OFF
  Enforcement: OFF
  Inode: N/A
Project quota state on /home （/dev/mapper/centos-home）
 Accounting: ON
  Enforcement: OFF
  Inode: N/A
Blocks grace time: [7 days 00:00:30]
Inodes grace time: [7 days 00:00:30]
Realtime Blocks grace time: [7 days 00:00:30]

[root@study ~]# dd if=/dev/zero of=/home/myquota/123.img bs=1M count=520
520+0 records in
520+0 records out  # 见鬼！竟然没有任何错误发生了！
545259520 Bytes （545 MB） copied, 0.308407 s, 180 MB/s

[root@study ~]# xfs_quota -x -c "report -pbh" /home
Project quota on /home （/dev/mapper/centos-home）
                        Blocks
Project ID       Used   Soft   Hard Warn/Grace
---------- ---------------------------------
myquotaproject   520M   450M   500M  00 [-none-]
# 其实，还真的有超过耶！只是因为 disable 的关系，所以没有强制限制住就是了！

[root@study ~]# xfs_quota -x -c "enable -up" /home  # 重新启动 quota 限制
[root@study ~]# dd if=/dev/zero of=/home/myquota/123.img bs=1M count=520
dd: error writing ‘/home/myquota/123.img’: No space left on device
# 又开始有限制！这就是 enable/disable 的相关对应功能喔！暂时关闭/启动用的！

# 完全关闭 quota 的限制行为吧！同时取消 project 的功能试看看！
[root@study ~]# xfs_quota -x -c "off -up" /home
[root@study ~]# xfs_quota -x -c "enable -up" /home
XFS_QUOTAON: Function not implemented
# 您瞧瞧！没有办法重新启动！因为已经完全的关闭了 quota 的功能！所以得要 umouont/mount 才行！

[root@study ~]# umount /home; mount -a
# 这个时候使用 report 以及 state 时，管制限制的内容又重新回来了！好！来瞧瞧如何移除project

[root@study ~]# xfs_quota -x -c "off -up" /home
[root@study ~]# xfs_quota -x -c "remove -p" /home
[root@study ~]# umount /home; mount -a
[root@study ~]# xfs_quota -x -c "report -phb" /home
Project quota on /home （/dev/mapper/centos-home）
                        Blocks
Project ID       Used   Soft   Hard Warn/Grace
---------- ---------------------------------
myquotaproject   500M      0      0  00 [------]
# 嘿嘿！全部归零！就是“移除”所有限制值的意思！
```

请注意上表中最后一个练习，那个 remove -p 是“移除所有的 project 控制列表”的意思！也就是说，如果你有在 /home 设置多个 project 的限制， 那么 remove 会删的一个也不留喔！如果想要回复设置值，那...只能一个一个重新设置回去了！没有好办法！

上面就是 XFS 文件系统的简易 quota 处理流程～那如果你是使用 EXT 家族呢？能不能使用 quota 呢？除了参考上一版的文件之外，鸟哥这里也列出相关的参考指令/设置文件给你对照参考！ 没学过的可以看看流程，有学过的可以对照了解！ ^_^

设置流程项目	**XFS**文件系统	**EXT**家族
/etc/fstab参数设置	usrquota/grpquota/prjquota	usrquota/grpquota
quota 配置文件	不需要	quotacheck
设置用户/群组限制值	xfs_quota -x -c "limit..."	edquota 或 setquota
设置 grace time	xfs_quota -x -c "timer..."	edquota
设置目录限制值	xfs_quota -x -c "limit..."	无
观察报告	xfs_quota -x -c "report..."	repquota 或 quota
启动与关闭 quota 限制	xfs_quota -x -c "[disable|enable]..."	quotaoff, quotaon
发送警告信给用户	目前版本尚未支持	warnquota

14.1.8 不更动既有系统的 quota 实例

想一想，如果你的主机原先没有想到要设置成为邮件主机，所以并没有规划将邮件信箱所在的 /var/spool/mail/ 目录独立成为一个 partition ，然后目前你的主机已经没有办法新增或分区出任何新的分区了。那我们知道 quota 的支持与文件系统有关， 所以并无法跨文件系统来设计 quota 的 project 功能啊！因此，你是否就无法针对 mail 的使用量给予 quota 的限制呢？

此外，如果你想要让使用者的邮件信箱与主文件夹的总体磁盘使用量为固定，那又该如何是好？ 由于 /home 及 /var/spool/mail 根本不可能是同一个 filesystem （除非是都不分区，使用根目录，才有可能整合在一起）， 所以，该如何进行这样的 quota 限制呢？

其实没有那么难啦！既然 quota 是针对 filesystem 来进行限制，假设你又已经有 /home 这个独立的分区了，那么你只要：

1. 将 /var/spool/mail 这个目录完整的移动到 /home 下面；
2. 利用 ln -s /home/mail /var/spool/mail 来创建链接数据；
3. 将 /home 进行 quota 限额设置

只要这样的一个小步骤，嘿嘿！您家主机的邮件就有一定的限额啰！当然啰！您也可以依据不同的使用者与群组来设置 quota 然后同样的以上面的方式来进行 link 的动作！嘿嘿嘿！就有不同的限额针对不同的使用者提出啰！很方便吧！^_^



Tips 朋友们需要注意的是，由于目前新的 distributions 大多有使用 SELinux 的机制， 因此你要进行如同上面的目录搬移时，在许多情况下可能会有使用上的限制喔！或许你得要先暂时关闭 SELinux 才能测试， 也或许你得要自行修改 SELinux 的规则才行喔！

14.2 软件磁盘阵列 （Software RAID）

在过去鸟哥还年轻的时代，我们能使用的硬盘容量都不大，几十 GB 的容量就是大硬盘了！ 但是某些情况下，我们需要很大容量的储存空间， 例如鸟哥在跑的空气品质模式所输出的数据文件一个案例通常需要好几 GB ，连续跑个几个案例，磁盘容量就不够用了。 此时我该如何是好？其实可以通过一种储存机制，称为磁盘阵列 （RAID） 的就是了。这种机制的功能是什么？他有哪些等级？什么是硬件、软件磁盘阵列？Linux 支持什么样的软件磁盘阵列？ 下面就让我们来谈谈！

14.2.1 什么是 RAID

磁盘阵列全名是“ Redundant Arrays of Inexpensive Disks, RAID ”，英翻中的意思是：容错式廉价磁盘阵列。 RAID 可以通过一个技术（软件或硬件），将多个较小的磁盘整合成为一个较大的磁盘设备； 而这个较大的磁盘功能可不止是储存而已，他还具有数据保护的功能呢。 整个 RAID 由于选择的等级 （level） 不同，而使得整合后的磁盘具有不同的功能，基本常见的 level 有这几种[2]：

- RAID-0 （等量模式, stripe） ：性能最佳

这种模式如果使用相同型号与容量的磁盘来组成时，效果较佳。这种模式的 RAID 会将磁盘先切出等量的区块 （名为chunk，一般可设置 4K~1M 之间） ， 然后当一个文件要写入 RAID 时，该文件会依据 chunk 的大小切割好，之后再依序放到各个磁盘里面去。由于每个磁盘会交错的存放数据， 因此当你的数据要写入 RAID 时，数据会被等量的放置在各个磁盘上面。 举例来说，你有两颗磁盘组成 RAID-0 ， 当你有 100MB 的数据要写入时，每个磁盘会各被分配到 50MB 的储存量。RAID-0 的示意图如下所示：

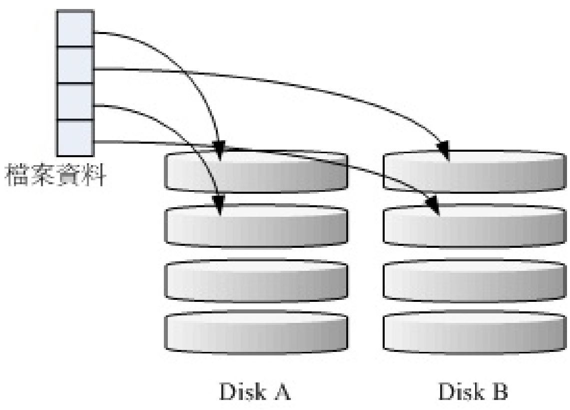


图14.2.1、RAID-0 的磁盘写入示意图

上图的意思是，在组成 RAID-0 时，每颗磁盘 （Disk A 与 Disk B） 都会先被区隔成为小区块 （chunk）。 当有数据要写入 RAID 时，数据会先被切割成符合小区块的大小，然后再依序一个一个的放置到不同的磁盘去。 由于数据已经先被切割并且依序放置到不同的磁盘上面，因此每颗磁盘所负责的数据量都降低了！照这样的情况来看， 越多颗磁盘组成的 RAID-0 性能

会越好，因为每颗负责的数据量就更低了！ 这表示我的数据可以分散让多颗磁盘来储存，当然性能会变的更好啊！此外，磁盘总容量也变大了！ 因为每颗磁盘的容量最终会加总成为 RAID-0 的总容量喔！

只是使用此等级你必须要自行负担数据损毁的风险，由上图我们知道文件是被切割成为适合每颗磁盘分区区块的大小， 然后再依序放置到各个磁盘中。想一想，如果某一颗磁盘损毁了，那么文件数据将缺一块，此时这个文件就损毁了。 由于每个文件都是这样存放的，因此 RAID-0 只要有任何一颗磁盘损毁，在 RAID 上面的所有数据都会遗失而无法读取。

另外，如果使用不同容量的磁盘来组成 RAID-0 时，由于数据是一直等量的依序放置到不同磁盘中，当小容量磁盘的区块被用完了， 那么所有的数据都将被写入到最大的那颗磁盘去。举例来说，我用 200G 与 500G 组成 RAID-0 ， 那么最初的 400GB 数据可同时写入两颗磁盘（各消耗 200G 的容量），后来再加入的数据就只能写入 500G 的那颗磁盘中了。 此时的性能就变差了，因为只剩下一颗可以存放数据嘛！

- RAID-1 （映射模式, mirror）：完整备份

这种模式也是需要相同的磁盘容量的，最好是一模一样的磁盘啦！如果是不同容量的磁盘组成 RAID-1 时，那么总容量将以最小的那一颗磁盘为主！这种模式主要是“让同一份数据，完整的保存在两颗磁盘上头”。举例来说，如果我有一个 100MB 的文件，且我仅有两颗磁盘组成 RAID-1 时， 那么这两颗磁盘将会同步写入 100MB 到他们的储存空间去。 因此，整体 RAID 的容量几乎少了 50%。由于两颗硬盘内容一模一样，好像镜子映照出来一样， 所以我们也称他为 mirror 模式啰～

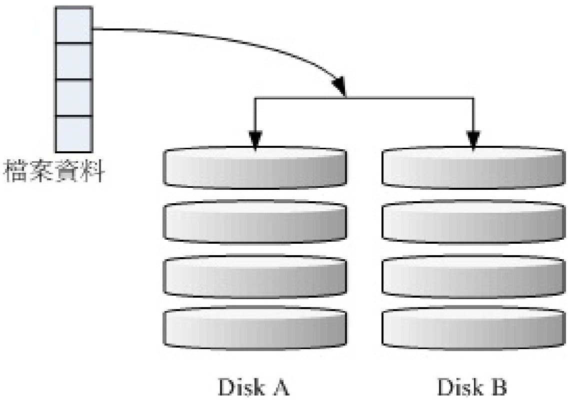


图14.2.2、RAID-1 的磁盘写入示意图

如上图所示，一份数据传送到 RAID-1 之后会被分为两股，并分别写入到各个磁盘里头去。 由于同一份数据会被分别写入到其他不同磁盘，因此如果要写入 100MB 时，数据传送到 I/O 总线后会被复制多份到各个磁盘， 结果就是数据量感觉变大了！因此在大量写入 RAID-1 的情况下，写入的性能可能会变的非常差 （因为我们只有一个南桥啊！）。 好在如果你使用的是硬件 RAID （磁盘阵列卡） 时，磁盘阵列卡会主动的复制一份而不使用系统的 I/O 总线，性能方面则还可以。 如果使用软件磁盘阵列，可能性能就不好了。

由于两颗磁盘内的数据一模一样，所以任何一颗硬盘损毁时，你的数据还是可以完整的保留下来的！ 所以我们可以说， RAID-1 最大的优点大概就在于数据的备份吧！不过由于磁盘容量有一半用在备份， 因此总容量会是全部磁盘容量的一半而已。虽然 RAID-1 的写入性能不

佳，不过读取的性能则还可以啦！这是因为数据有两份在不同的磁盘上面，如果多个 processes 在读取同一笔数据时， RAID 会自行取得最佳的读取平衡。

- RAID 1+0，RAID 0+1

RAID-0 的性能佳但是数据不安全，RAID-1 的数据安全但是性能不佳，那么能不能将这两者整合起来设置 RAID 呢？ 可以啊！那就是 RAID 1+0 或 RAID 0+1。所谓的 RAID 1+0 就是：（1）先让两颗磁盘组成 RAID 1，并且这样的设置共有两组；（2）将这两组 RAID 1 再组成一组 RAID 0。这就是 RAID 1+0 啰！反过来说，RAID 0+1 就是先组成 RAID-0 再组成 RAID-1 的意思。

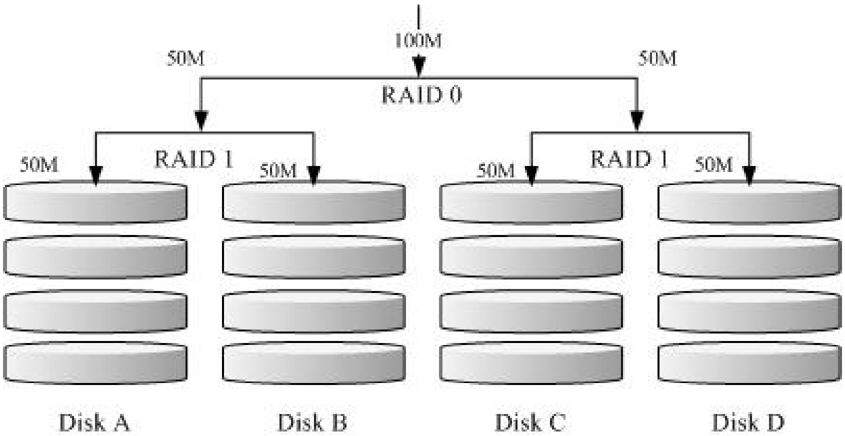


图14.2.3、RAID-1+0 的磁盘写入示意图

如上图所示，Disk A + Disk B 组成第一组 RAID 1，Disk C + Disk D 组成第二组 RAID 1， 然后这两组再整合成为一组 RAID 0。如果我有 100MB 的数据要写入，则由于 RAID 0 的关系， 两组 RAID 1 都会写入 50MB，又由于 RAID 1 的关系，因此每颗磁盘就会写入 50MB 而已。 如此一来不论哪一组 RAID 1 的磁盘损毁，由于是 RAID 1 的图像数据，因此就不会有任何问题发生了！这也是目前储存设备厂商最推荐的方法！



Tips 为何会推荐 RAID 1+0 呢？想像你有 20 颗磁盘组成的系统，每两颗组成一个 RAID1，因此你就有总共 10组可以自己复原的系统了！ 然后这 10组再组成一个新的 RAID0，速度立刻拉升 10倍了！同时要注意，因为每组 RAID1 是个别独立存在的，因此任何一颗磁盘损毁，数据都是从另一颗磁盘直接复制过来重建，并不像 RAID5/RAID6 必须要整组 RAID 的磁盘共同重建一颗独立的磁盘系统！性能上差非常多！ 而且 RAID 1 与 RAID 0 是不需要经过计算的（striping）！读写性能也比其他的 RAID 等级好太多了！

- RAID 5：性能与数据备份的均衡考虑

RAID-5 至少需要三颗以上的磁盘才能够组成这种类型的磁盘阵列。这种磁盘阵列的数据写入有点类似 RAID-0 ，不过每个循环的写入过程中 （striping），在每颗磁盘还加入一个同位检查数据 （Parity） ，这个数据会记录其他磁盘的备份数据， 用于当有磁盘损毁时的救援。RAID-5 读写的情况有点像下面这样：

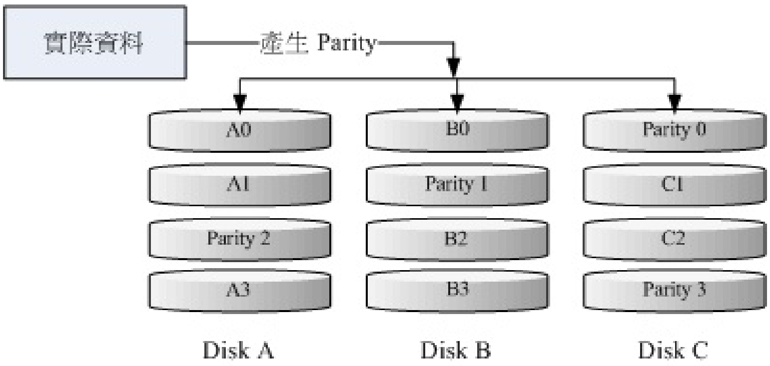


图14.2.4、RAID-5 的磁盘写入示意图

如上图所示，每个循环写入时，都会有部分的同位检查码 （parity） 被记录起来，并且记录的同位检查码每次都记录在不同的磁盘， 因此，任何一个磁盘损毁时都能够借由其他磁盘的检查码来重建原本磁盘内的数据喔！不过需要注意的是， 由于有同位检查码，因此 RAID 5 的总容量会是整体磁盘数量减一颗。以上图为例， 原本的 3 颗磁盘只会剩下 （3-1）=2 颗磁盘的容量。而且当损毁的磁盘数量大于等于两颗时，这整组 RAID 5 的数据就损毁了。 因为 RAID 5 默认仅能支持一颗磁盘的损毁情况。

在读写性能的比较上，读取的性能还不赖！与 RAID-0 有的比！不过写的性能就不见得能够增加很多！ 这是因为要写入 RAID 5 的数据还得要经过计算同位检查码 （parity） 的关系。由于加上这个计算的动作， 所以写入的性能与系统的硬件关系较大！尤其当使用软件磁盘阵列时，同位检查码是通过 CPU 去计算而非专职的磁盘阵列卡， 因此性能方面还需要评估。

另外，由于 RAID 5 仅能支持一颗磁盘的损毁，因此近来还有发展出另外一种等级，就是 RAID 6 ，这个 RAID 6 则使用两颗磁盘的容量作为 parity 的储存，因此整体的磁盘容量就会少两颗，但是允许出错的磁盘数量就可以达到两颗了！ 也就是在 RAID 6 的情况下，同时两颗磁盘损毁时，数据还是可以救回来！

- Spare Disk：预备磁盘的功能：

当磁盘阵列的磁盘损毁时，就得要将坏掉的磁盘拔除，然后换一颗新的磁盘。换成新磁盘并且顺利启动磁盘阵列后， 磁盘阵列就会开始主动的重建 （rebuild） 原本坏掉的那颗磁盘数据到新的磁盘上！然后你磁盘阵列上面的数据就复原了！ 这就是磁盘阵列的优点。不过，我们还是得要动手拔插硬盘，除非你的系统有支持热拔插，否则通常得要关机才能这么做。

为了让系统可以实时的在坏掉硬盘时主动的重建，因此就需要预备磁盘 （spare disk） 的辅助。 所谓的 spare disk 就是一颗或多颗没有包含在原本磁盘阵列等级中的磁盘，这颗磁盘平时并不会被磁盘阵列所使用， 当磁盘阵列有任何磁盘损毁时，则这颗 spare disk 会被主动的

拉进磁盘阵列中，并将坏掉的那颗硬盘移出磁盘阵列！ 然后立即重建数据系统。如此你的系统则可以永保安康啊！若你的磁盘阵列有支持热拔插那就更完美了！ 直接将坏掉的那颗磁盘拔除换一颗新的，再将那颗新的设置成为 spare disk ，就完成了！

举例来说，鸟哥之前所待的研究室有一个磁盘阵列可允许 16 颗磁盘的数量，不过我们只安装了 10 颗磁盘作为 RAID 5。 每颗磁盘的容量为 250GB，我们用了一颗磁盘作为 spare disk ，并将其他的 9 颗设置为一个 RAID 5， 因此这个磁盘阵列的总容量为： (9-1)*250G=2000G。运行了一两年后真的有一颗磁盘坏掉了，我们后来看灯号才发现！ 不过对系统没有影响呢！因为 spare disk 主动的加入支持，坏掉的那颗拔掉换颗新的，并重新设置成为 spare 后， 系统内的数据还是完整无缺的！嘿嘿！真不错！

- 磁盘阵列的优点

说的口沫横飞，重点在哪里呢？其实你的系统如果需要磁盘阵列的话，其实重点在于：

1. 数据安全与可靠性：指的并非网络信息安全，而是当硬件 (指磁盘) 损毁时，数据是否还能够安全的救援或使用之意；
2. 读写性能：例如 RAID 0 可以加强读写性能，让你的系统 I/O 部分得以改善；
3. 容量：可以让多颗磁盘组合起来，故单一文件系统可以有相当大的容量。

尤其数据的可靠性与完整性更是使用 RAID 的考虑重点！毕竟硬件坏掉换掉就好了，软件数据损毁那可不是闹着玩的！ 所以企业界为何需要大量的 RAID 来做为文件系统的硬件基准，现在您有点了解了吧？那依据这三个重点，我们来列表看看上面几个重要的 RAID 等级各有哪些优点吧！假设有 n 颗磁盘组成的 RAID 设置喔！

项目	RAID0	RAID1	RAID10	RAID5	RAID6
最少磁盘数	2	2	4	3	4
最大容错磁盘数(1)	无	n-1	n/2	1	2
数据安全性(1)	完全没有	最佳	最佳	好	比 RAID5 好
理论写入性能(2)	n	1	n/2	<n-1	<n-2
理论读出性能(2)	n	n	n	<n-1	<n-2
可用容量(3)	n	1	n/2	n-1	n-2
一般应用	强调性能但数据不重要的环境	数据与备份	服务器、云系统常用	数据与备份	数据与备份

因为 RAID5, RAID6 读写都需要经过 parity 的计算机制，因此读/写性能都不会刚好满足于使用的磁盘数量喔！

另外，根据使用的情况不同，一般推荐的磁盘阵列等级也不太一样。以鸟哥为例，在鸟哥的跑空气品质模式之后的输出数据，动辄几百 GB 的单一大文件数据， 这些情况鸟哥会选择放在 RAID6 的阵列环境下，这是考虑到数据保全与总容量的应用，因为 RAID 6 的性能已经足以应付模式读入所需的环境。

近年来鸟哥也比较积极在作一些云程序环境的设计，在云环境下，确保每个虚拟机能够快速的反应以及提供数据保全是最重要的部份！ 因此性能方面比较弱的 RAID5/RAID6 是不考虑的，总结来说，大概就剩下 RAID10 能够满足云环境的性能需求了。在某些更特别的环境下， 如果搭配 SSD 那才更具有性能上的优势哩！

14.2.2 software, hardware RAID

为何磁盘阵列又分为硬件与软件呢？所谓的硬件磁盘阵列 （hardware RAID） 是通过磁盘阵列卡来达成阵列的目的。 磁盘阵列卡上面有一块专门的芯片在处理 RAID 的任务，因此在性能方面会比较好。在很多任务 （例如 RAID 5 的同位检查码计算） 磁盘阵列并不会重复消耗原本系统的 I/O 总线，理论上性能会较佳。此外目前一般的中高阶磁盘阵列卡都支持热拔插，亦即在不关机的情况下抽换损坏的磁盘，对于系统的复原与数据的可靠性方面非常的好用。

不过一块好的磁盘阵列卡动不动就上万元台币，便宜的在主板上面“附赠”的磁盘阵列功能可能又不支持某些高阶功能， 例如低阶主板若有磁盘阵列芯片，通常仅支持到 RAID0 与 RAID1 ，鸟哥喜欢的 RAID6 并没有支持。 此外，操作系统也必须要拥有磁盘阵列卡的驱动程序，才能够正确的捉到磁盘阵列所产生的磁盘机！

由于磁盘阵列有很多优秀的功能，然而硬件磁盘阵列卡偏偏又贵的很～因此就有发展出利用软件来仿真磁盘阵列的功能， 这就是所谓的软件磁盘阵列 （software RAID）。软件磁盘阵列主要是通过软件来仿真阵列的任务， 因此会损耗较多的系统资源，比如说 CPU 的运算与 I/O 总线的资源等。不过目前我们的个人计算机实在已经非常快速了， 因此以前的速度限制现在已经不存在！所以我们可以来玩一玩软件磁盘阵列！

我们的 CentOS 提供的软件磁盘阵列为 mdadm 这套软件，这套软件会以 partition 或 disk 为磁盘的单位，也就是说，你不需要两颗以上的磁盘，只要有两个以上的分区 （partition） 就能够设计你的磁盘阵列了。此外， mdadm 支持刚刚我们前面提到的 RAID0/RAID1/RAID5/spare disk 等！ 而且提供的管理机制还可以达到类似热拔插的功能，可以线上 （文件系统正常使用） 进行分区的抽换！ 使用上也非常的方便呢！

另外你必须要知道的是，硬件磁盘阵列在 Linux 下面看起来就是一颗实际的大磁盘，因此硬件磁盘阵列的设备文件名为 /dev/sd[a-p] ，因为使用到 SCSI 的模块之故。至于软件磁盘阵列则是系统仿真的，因此使用的设备文件名是系统的设备文件， 文件名为 /dev/md0, /dev/md1...，两者的设备文件名并不相同！不要搞混了喔！因为很多朋友常常觉得奇怪， 怎么他的 RAID 文件名跟我们这里测试的软件 RAID 文件名不同，所以这里特别强调说明喔！



Tips Intel 的南桥附赠的磁盘阵列功能，在 windows 下面似乎是完整的磁盘阵列，但是在 Linux 下面则被视为是软件磁盘阵列的一种！ 因此如果你有设置过 Intel 的南桥芯片磁盘阵列，那在 Linux 下面反而还会是 /dev/md126, /dev/md127 等等设备文件名， 而他的分区竟然是 /dev/md126p1, /dev/md126p2... 之类的喔！比较特别，所以这里加强说明！

14.2.3 软件磁盘阵列的设置

软件磁盘阵列的设置很简单呢！简单到让你很想笑喔！因为你只要使用一个指令即可！那就是 mdadm 这个指令。 这个指令在创建 RAID 的语法有点像这样：

```
[root@study ~]# mdadm --detail /dev/md0
[root@study ~]# mdadm --create /dev/md[0-9] --auto=yes --level=[015] --chunk=NK \
&gt; --raid-devices=N --spare-devices=N /dev/sdx /dev/hdx...
选项与参数：
--create            ：为创建 RAID 的选项；
--auto=yes          ：决定创建后面接的软件磁盘阵列设备，亦即 /dev/md0, /dev/md1...
--chunk=Nk          ：决定这个设备的 chunk 大小，也可以当成 stripe 大小，一般是 64K 或 512K。
--raid-devices=N    ：使用几个磁盘 （partition） 作为磁盘阵列的设备
--spare-devices=N   ：使用几个磁盘作为备用 （spare） 设备
--level=[015]       ：设置这组磁盘阵列的等级。支持很多，不过建议只要用 0, 1, 5 即可
--detail            ：后面所接的那个磁盘阵列设备的详细信息
```

上面的语法中，最后面会接许多的设备文件名，这些设备文件名可以是整颗磁盘，例如 /dev/sdb ， 也可以是分区，例如 /dev/sdb1 之类。不过，这些设备文件名的总数必须要等于 --raid-devices 与 --spare-devices 的个数总和才行！鸟哥利用我的测试机来创建一个 RAID 5 的软件磁盘阵列给您瞧瞧！ 下面是鸟哥希望做成的 RAID 5 环境：

- 利用 4 个 partition 组成 RAID 5；
- 每个 partition 约为 1GB 大小，需确定每个 partition 一样大较佳；
- 利用 1 个 partition 设置为 spare disk
- chunk 设置为 256K 这么大即可！
- 这个 spare disk 的大小与其他 RAID 所需 partition 一样大！
- 将此 RAID 5 设备挂载到 /srv/raid 目录下

最终我需要 5 个 1GB 的 partition。在鸟哥的测试机中，根据前面的章节实做下来，包括课后的情境仿真题目，目前应该还有 8GB 可供利用！ 因此就利用这部测试机的 /dev/vda 切出 5 个 1G 的分区。实际的流程鸟哥就不一一展示了，自己通过 gdisk /dev/vda 实作一下！ 最终这部测试机的结果应该如下所示：

```
[root@study ~]# gdisk -l /dev/vda
Number  Start （sector）    End （sector）  Size       Code  Name
   1            2048            6143   2.0 MiB     EF02
   2            6144         2103295   1024.0 MiB  0700
   3         2103296        65026047   30.0 GiB    8E00
   4        65026048        67123199   1024.0 MiB  8300  Linux filesystem
 5        67123200        69220351   1024.0 MiB  FD00  Linux RAID
   6        69220352        71317503   1024.0 MiB  FD00  Linux RAID
   7        71317504        73414655   1024.0 MiB  FD00  Linux RAID
   8        73414656        75511807   1024.0 MiB  FD00  Linux RAID
   9        75511808        77608959   1024.0 MiB  FD00  Linux RAID
# 上面特殊字体的部份就是我们需要的那 5 个 partition 啰！注意注意！

[root@study ~]# lsblk
NAME            MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
vda             252:0    0   40G  0 disk
&#124;-vda1          252:1    0    2M  0 part
&#124;-vda2          252:2    0    1G  0 part /boot
&#124;-vda3          252:3    0   30G  0 part
&#124; &#124;-centos-root 253:0    0   10G  0 lvm  /
&#124; &#124;-centos-swap 253:1    0    1G  0 lvm  [SWAP]
&#124; `-centos-home 253:2    0    5G  0 lvm  /home
&#124;-vda4          252:4    0    1G  0 part /srv/myproject
&#124;-vda5          252:5    0    1G  0 part
&#124;-vda6          252:6    0    1G  0 part
&#124;-vda7          252:7    0    1G  0 part
&#124;-vda8          252:8    0    1G  0 part
`-vda9          252:9    0    1G  0 part
```

- 以 mdadm 创建 RAID

接下来就简单啦！通过 mdadm 来创建磁盘阵列先！

```
[root@study ~]# mdadm --create /dev/md0 --auto=yes --level=5 --chunk=256K \
&gt;  --raid-devices=4 --spare-devices=1 /dev/vda{5,6,7,8,9}
mdadm: /dev/vda5 appears to contain an ext2fs file system
       size=1048576K  mtime=Thu Jun 25 00:35:01 2015   # 某些时刻会出现这个东西！没关系的！
Continue creating array? y
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
# 详细的参数说明请回去前面看看啰！这里我通过 {} 将重复的项目简化！
# 此外，因为鸟哥这个系统经常在创建测试的环境，因此系统可能会抓到之前的 filesystem
# 所以就会出现如上前两行的讯息！那没关系的！直接按下 y 即可删除旧系统

[root@study ~]# mdadm --detail /dev/md0
/dev/md0:                                           # RAID 的设备文件名
        Version : 1.2
  Creation Time : Mon Jul 27 15:17:20 2015          # 创建 RAID 的时间
     Raid Level : raid5                             # 这就是 RAID5 等级！
     Array Size : 3142656 (3.00 GiB 3.22 GB)         # 整组 RAID 的可用容量
  Used Dev Size : 1047552 (1023.17 MiB 1072.69 MB)   # 每颗磁盘（设备）的容量
   Raid Devices : 4                                 # 组成 RAID 的磁盘数量
  Total Devices : 5                                 # 包括 spare 的总磁盘数
    Persistence : Superblock is persistent

    Update Time : Mon Jul 27 15:17:31 2015
          State : clean                             # 目前这个磁盘阵列的使用状态
 Active Devices : 4                                 # 启动（active）的设备数量
Working Devices : 5                                 # 目前使用于此阵列的设备数
 Failed Devices : 0                                 # 损坏的设备数
  Spare Devices : 1                                 # 预备磁盘的数量

         Layout : left-symmetric
     Chunk Size : 256K                              # 就是 chunk 的小区块容量

           Name : study.centos.vbird:0  (local to host study.centos.vbird)
           UUID : 2256da5f:4870775e:cf2fe320:4dfabbc6
         Events : 18

    Number   Major   Minor   RaidDevice State
       0     252        5        0      active sync   /dev/vda5
       1     252        6        1      active sync   /dev/vda6
       2     252        7        2      active sync   /dev/vda7
       5     252        8        3      active sync   /dev/vda8

       4     252        9        -      spare   /dev/vda9
# 最后五行就是这五个设备目前的情况，包括四个 active sync 一个 spare ！
# 至于 RaidDevice  指的则是此 RAID 内的磁盘顺序
```

由于磁盘阵列的创建需要一些时间，所以你最好等待数分钟后再使用“ mdadm --detail /dev/md0 ”去查阅你的磁盘阵列详细信息！ 否则有可能看到某些磁盘正在“spare rebuilding”之类的创建字样！通过上面的指令， 你就能够创建一个 RAID5 且含有一颗 spare disk 的磁盘阵列啰！非常简单吧！ 除了指令之外，你也可以查阅如下的文件来看看系统软件磁盘阵列的情况：

```
[root@study ~]# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 vda8[5] vda9[4](S) vda7[2] vda6[1] vda5[0]              &lt;==第一行
      3142656 blocks super 1.2 level 5, 256k chunk, algorithm 2 [4/4] [UUUU] &lt;==第二行

unused devices: &lt;none&gt;
```

上述的数据比较重要的在特别指出的第一行与第二行部分[3]：

- 第一行部分：指出 md0 为 raid5 ，且使用了 vda8, vda7, vda6, vda5 等四颗磁盘设备。每个设备后面的中括号 [] 内的数字为此磁盘在 RAID 中的顺序 （RaidDevice）；至于 vda9 后面的 [S] 则代表 vda9 为 spare 之意。
- 第二行：此磁盘阵列拥有 3142656 个block（每个 block 单位为 1K），所以总容量约为 3GB， 使用 RAID 5 等级，写入磁盘的小区块 （chunk） 大小为 256K，使用 algorithm 2 磁盘阵列演算法。 [m/n] 代表此阵列需要 m 个设备，且 n 个设备正常运行。因此本 md0 需要 4 个设备且这 4 个设备均正常运行。 后面的 [UUUU] 代表的是四个所需的设备 （就是 [m/n] 里面的 m） 的启动情况，U 代表正常运行，若为 _ 则代表不正常。

这两种方法都可以知道目前的磁盘阵列状态啦！

- 格式化与挂载使用 RAID

接下来就是开始使用格式化工具啦！这部分就需要注意喔！因为涉及到 xfs 文件系统的优化！还记得第七章的内容吧？我们这里的参数为：

- srtipe （chunk） 容量为 256K，所以 su=256k
- 共有 4 颗组成 RAID5 ，因此容量少一颗，所以 sw=3 喔！
- 由上面两项计算出数据宽度为： 256K*3=768k

所以整体来说，要优化这个 XFS 文件系统就变成这样：

```
[root@study ~]# mkfs.xfs -f -d su=256k,sw=3 -r extsize=768k /dev/md0
# 有趣吧！是 /dev/md0 做为设备被格式化呢！

[root@study ~]# mkdir /srv/raid
[root@study ~]# mount /dev/md0 /srv/raid
[root@study ~]# df -Th /srv/raid
Filesystem     Type  Size  Used Avail Use% Mounted on
/dev/md0       xfs   3.0G   33M  3.0G   2% /srv/raid
# 看吧！多了一个 /dev/md0 的设备，而且真的可以让你使用呢！还不赖！
```

14.2.4 仿真 RAID 错误的救援模式

俗话说“天有不测风云、人有旦夕祸福”，谁也不知道你的磁盘阵列内的设备啥时会出差错，因此， 了解一下软件磁盘阵列的救援还是必须的！下面我们就来玩一玩救援的机制吧！首先来了解一下 mdadm 这方面的语法：

```
[root@study ~]# mdadm --manage /dev/md[0-9] [--add 设备] [--remove 设备] [--fail 设备]
选项与参数：
--add     ：会将后面的设备加入到这个 md 中！
--remove ：会将后面的设备由这个 md 中移除
--fail    ：会将后面的设备设置成为出错的状态
```

- 设置磁盘为错误 （fault）

首先，我们来处理一下，该如何让一个磁盘变成错误，然后让 spare disk 自动的开始重建系统呢？

```
# 0\. 先复制一些东西到 /srv/raid 去，假设这个 RAID 已经在使用了
[root@study ~]# cp -a /etc /var/log /srv/raid
[root@study ~]# df -Th /srv/raid ; du -sm /srv/raid/*
Filesystem     Type  Size  Used Avail Use% Mounted on
/dev/md0       xfs   3.0G  144M  2.9G   5% /srv/raid
28      /srv/raid/etc  &lt;==看吧！确实有数据在里面喔！
51      /srv/raid/log

# 1\. 假设 /dev/vda7 这个设备出错了！实际仿真的方式：
[root@study ~]# mdadm --manage /dev/md0 --fail /dev/vda7
mdadm: set /dev/vda7 faulty in /dev/md0      # 设置成为错误的设备啰！
/dev/md0:
.....（中间省略）.....
    Update Time : Mon Jul 27 15:32:50 2015
          State : clean, degraded, recovering
 Active Devices : 3
Working Devices : 4
 Failed Devices : 1      &lt;==出错的磁盘有一个！
  Spare Devices : 1
.....（中间省略）.....

    Number   Major   Minor   RaidDevice State
       0     252        5        0      active sync   /dev/vda5
       1     252        6        1      active sync   /dev/vda6
       4     252        9        2      spare rebuilding   /dev/vda9
       5     252        8        3      active sync   /dev/vda8

       2     252        7        -      faulty   /dev/vda7
# 看到没！这的动作要快做才会看到！ /dev/vda9 启动了而 /dev/vda7 死掉了
```

上面的画面你得要快速的连续输入那些 mdadm 的指令才看的到！因为你的 RAID 5 正在重建系统！ 若你等待一段时间再输入后面的观察指令，则会看到如下的画面了：

```
# 2\. 已经借由 spare disk 重建完毕的 RAID 5 情况
[root@study ~]# mdadm --detail /dev/md0
....（前面省略）....
    Number   Major   Minor   RaidDevice State
       0     252        5        0      active sync   /dev/vda5
       1     252        6        1      active sync   /dev/vda6
       4     252        9        2      active sync   /dev/vda9
       5     252        8        3      active sync   /dev/vda8

       2     252        7        -      faulty   /dev/vda7
```

看吧！又恢复正常了！真好！我们的 /srv/raid 文件系统是完整的！并不需要卸载！很棒吧！

- 将出错的磁盘移除并加入新磁盘

因为我们的系统那个 /dev/vda7 实际上没有坏掉啊！只是用来仿真而已啊！因此，如果有新的磁盘要替换，其实替换的名称会一样啊！ 也就是我们需要：

1. 先从 /dev/md0 阵列中移除 /dev/vda7 这颗“磁盘”
2. 整个 Linux 系统关机，拔出 /dev/vda7 这颗“磁盘”，并安装上新的 /dev/vda7 “磁盘”，之后开机
3. 将新的 /dev/vda7 放入 /dev/md0 阵列当中！

```
# 3\. 拔除“旧的”/dev/vda7 磁盘
[root@study ~]# mdadm --manage /dev/md0 --remove /dev/vda7
# 假设接下来你就进行了上面谈到的第 2, 3 个步骤，然后重新开机成功了！

# 4\. 安装“新的”/dev/vda7 磁盘
[root@study ~]# mdadm --manage /dev/md0 --add /dev/vda7
[root@study ~]# mdadm --detail /dev/md0
....（前面省略）....
    Number   Major   Minor   RaidDevice State
       0     252        5        0      active sync   /dev/vda5
       1     252        6        1      active sync   /dev/vda6
       4     252        9        2      active sync   /dev/vda9
       5     252        8        3      active sync   /dev/vda8

       6     252        7        -      spare   /dev/vda7
```

嘿嘿！你的磁盘阵列内的数据不但一直存在，而且你可以一直顺利的运行 /srv/raid 内的数据，即使 /dev/vda7 损毁了！然后通过管理的功能就能够加入新磁盘且拔除坏掉的磁盘！注意，这一切都是在上线（on-line）的情况下进行！所以，您说这样的咚咚好不好用啊！^_^

14.2.5 开机自动启动 RAID 并自动挂载

新的 distribution 大多会自己搜寻 /dev/md[0-9] 然后在开机的时候给予设置好所需要的功能。不过鸟哥还是建议你，修改一下配置文件吧！^_^。software RAID 也是有配置文件的，这个配置文件在 /etc/mdadm.conf ！这个配置文件内容很简单，你只要知道 /dev/md0 的 UUID 就能够设置这个文件啦！这里鸟哥仅介绍他最简单的语法：

```
[root@study ~]# mdadm --detail /dev/md0 &#124; grep -i uuid
           UUID : 2256da5f:4870775e:cf2fe320:4dfabbc6
# 后面那一串数据，就是这个设备向系统注册的 UUID 识别码！

# 开始设置 mdadm.conf
[root@study ~]# vim /etc/mdadm.conf
ARRAY /dev/md0 UUID=2256da5f:4870775e:cf2fe320:4dfabbc6
#     RAID设备      识别码内容

# 开始设置开机自动挂载并测试
[root@study ~]# blkid /dev/md0
/dev/md0: UUID="494cb3e1-5659-4efc-873d-d0758baec523" TYPE="xfs"

[root@study ~]# vim /etc/fstab
UUID=494cb3e1-5659-4efc-873d-d0758baec523  /srv/raid xfs defaults 0 0

[root@study ~]# umount /dev/md0; mount -a
[root@study ~]# df -Th /srv/raid
Filesystem     Type  Size  Used Avail Use% Mounted on
/dev/md0       xfs   3.0G  111M  2.9G   4% /srv/raid
# 你得确定可以顺利挂载，并且没有发生任何错误！
```

如果到这里都没有出现任何问题！接下来就请 reboot 你的系统并等待看看能否顺利的启动吧！^_^

14.2.6 关闭软件 RAID（重要！）

除非你未来就是要使用这颗 software RAID （/dev/md0），否则你势必要跟鸟哥一样，将这个 /dev/md0 关闭！ 因为他毕竟是我们在这个测试机上面的练习设备啊！为什么要关掉他呢？因为这个 /dev/md0 其实还是使用到我们系统的磁盘分区， 在鸟哥的例子里面就是 /dev/vda{5,6,7,8,9}，如果你只是将 /dev/md0 卸载，然后忘记将 RAID 关闭， 结果就是....未来你在重新分区 /dev/vdaX 时可能会出现一些莫名的错误状况啦！所以才需要关闭 software RAID 的步骤！ 那如何关闭呢？也是简单到爆炸！（请注意，确认你的 /dev/md0 确实不要用且要关闭了才进行下面的玩意儿）

```
# 1\. 先卸载且删除配置文件内与这个 /dev/md0 有关的设置：
[root@study ~]# umount /srv/raid
[root@study ~]# vim /etc/fstab
UUID=494cb3e1-5659-4efc-873d-d0758baec523  /srv/raid xfs defaults 0 0
# 将这一行删除掉！或者是注解掉也可以！

# 2\. 先覆盖掉 RAID 的 metadata 以及 XFS 的 superblock，才关闭 /dev/md0 的方法
[root@study ~]# dd if=/dev/zero of=/dev/md0 bs=1M count=50
[root@study ~]# mdadm --stop /dev/md0
mdadm: stopped /dev/md0  &lt;==不啰唆！这样就关闭了！
[root@study ~]# dd if=/dev/zero of=/dev/vda5 bs=1M count=10
[root@study ~]# dd if=/dev/zero of=/dev/vda6 bs=1M count=10
[root@study ~]# dd if=/dev/zero of=/dev/vda7 bs=1M count=10
[root@study ~]# dd if=/dev/zero of=/dev/vda8 bs=1M count=10
[root@study ~]# dd if=/dev/zero of=/dev/vda9 bs=1M count=10

[root@study ~]# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
unused devices: &lt;none&gt;  &lt;==看吧！确实不存在任何阵列设备！

[root@study ~]# vim /etc/mdadm.conf
#ARRAY /dev/md0 UUID=2256da5f:4870775e:cf2fe320:4dfabbc6
# 一样啦！删除他或是注解他！
```

你可能会问，鸟哥啊，为啥上面会有数个 dd 的指令啊？干麻？这是因为 RAID 的相关数据其实也会存一份在磁盘当中，因此，如果你只是将配置文件移除， 同时关闭了 RAID，但是分区并没有重新规划过，那么重新开机过后，系统还是会将这颗磁盘阵列创建起来，只是名称可能会变成 /dev/md127 就是了！ 因此，移除掉 Software RAID 时，上述的 dd 指令不要忘记！但是...千千万万不要 dd 到错误的磁盘～那可是会欲哭无泪耶～



Tips 在这个练习中，鸟哥使用同一颗磁盘进行软件 RAID 的实验。不过朋友们要注意的是，如果真的要实作软件磁盘阵列， 最好是由多颗不同的磁盘来组成较佳！因为这样才能够使用到不同磁盘的读写，性能才会好！ 而数据分配在不同的磁盘，当某颗磁盘损毁时数据才能够借由其他磁盘挽救回来！这点得特别留意呢！

14.3 逻辑卷轴管理员 （Logical Volume Manager）

想像一个情况，你在当初规划主机的时候将 /home 只给他 50G ，等到使用者众多之后导致这个 filesystem 不够大， 此时你能怎么作？多数的朋友都是这样：再加一颗新硬盘，然后重新分区、格式化，将 /home 的数据完整的复制过来， 然后将原本的 partition 卸载重新挂载新的 partition 。啊！好忙碌啊！若是第二次分区却给的容量太多！导致很多磁盘容量被浪费了！ 你想要将这个 partition 缩小时，又该如何作？将上述的流程再搞一遍！唉～烦死了，尤其复制很花时间ㄟ～有没有更简单的方法呢？ 有的！那就是我们这个小节要介绍的 LVM 这玩意儿！

LVM 的重点在于“可以弹性的调整 filesystem 的容量！”而并非在于性能与数据保全上面。 需要文件的读写性能或者是数据的可靠性，请参考前面的 RAID 小节。 LVM 可以整合多个实体 partition 在一起， 让这些 partitions 看起来就像是一个磁盘一样！而且，还可以在未来新增或移除其他的实体 partition 到这个 LVM 管理的磁盘当中。 如此一来，整个磁盘空间的使用上，实在是相当的具有弹性啊！ 既然 LVM 这么好用，那就让我们来瞧瞧这玩意吧！

14.3.1 什么是 LVM： PV, PE, VG, LV 的意义

LVM 的全名是 Logical Volume Manager，中文可以翻译作逻辑卷轴管理员。之所以称为“卷轴”可能是因为可以将 filesystem 像卷轴一样伸长或缩短之故吧！LVM 的作法是将几个实体的 partitions （或 disk） 通过软件组合成为一块看起来是独立的大磁盘 （VG） ，然后将这块大磁盘再经过分区成为可使用分区 （LV）， 最终就能够挂载使用了。但是为什么这样的系统可以进行 filesystem 的扩充或缩小呢？其实与一个称为 PE 的项目有关！ 下面我们就得要针对这几个项目来好好聊聊！

- Physical Volume, PV, 实体卷轴

我们实际的 partition （或 Disk） 需要调整系统识别码 （system ID） 成为 8e （LVM 的识别码），然后再经过 pvcreate 的指令将他转成 LVM 最底层的实体卷轴 （PV） ，之后才能够将这些 PV 加以利用！ 调整 system ID 的方是就是通过 gdisk 啦！

- Volume Group, VG, 卷轴群组

所谓的 LVM 大磁盘就是将许多 PV 整合成这个 VG 的东西就是啦！所以 VG 就是 LVM 组合起来的大磁盘！这么想就好了。 那么这个大磁盘最大可以到多少容量呢？这与下面要说明的 PE 以及 LVM 的格式版本有关喔～在默认的情况下， 使用 32位的 Linux 系统时，基本上 LV 最大仅能支持到 65534 个 PE 而已，若使用默认的 PE 为 4MB 的情况下， 最大容量则仅能达到约 256GB 而已～不过，这个问题在 64位的 Linux 系统上面已经不存在了！LV 几乎没有啥容量限制了！

- Physical Extent, PE, 实体范围区块

LVM 默认使用 4MB 的 PE 区块，而 LVM 的 LV 在 32 位系统上最多仅能含有 65534 个 PE（lvm1 的格式），因此默认的 LVM 的 LV 会有 4M*65534/（1024M/G）=256G。这个 PE 很有趣喔！他是整个 LVM 最小的储存区块，也就是说，其实我们的文件数据都是借由写入 PE 来处理的。简单的说，这个 PE 就有点像文件系统里面的 block 大小啦。 这样说应该就比较好理解了吧？所以调整 PE 会影响到 LVM 的最大容量喔！不过，在 CentOS 6.x 以后，由于直接使用 lvm2 的各项格式功能，以及系统转为 64 位，因此这个限制已经不存在了。

- Logical Volume, LV, 逻辑卷轴

最终的 VG 还会被切成 LV，这个 LV 就是最后可以被格式化使用的类似分区的咚咚了！那么 LV 是否可以随意指定大小呢？ 当然不可以！既然 PE 是整个 LVM 的最小储存单位，那么 LV 的大小就与在此 LV 内的 PE 总数有关。 为了方便使用者利用 LVM 来管理其系统，因此 LV 的设备文件名通常指定为“ /dev/vgname/lvname ”的样式！

此外，我们刚刚有谈到 LVM 可弹性的变更 filesystem 的容量，那是如何办到的？其实他就是通过“交换 PE ”来进行数据转换， 将原本 LV 内的 PE 移转到其他设备中以降低 LV 容量，或将其他设备的 PE 加到此 LV 中以加大容量！ VG、LV 与 PE 的关系有点像下图：

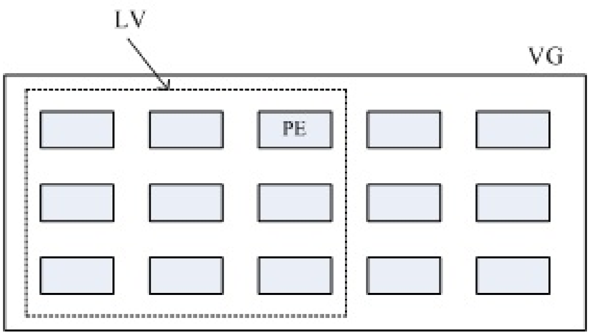
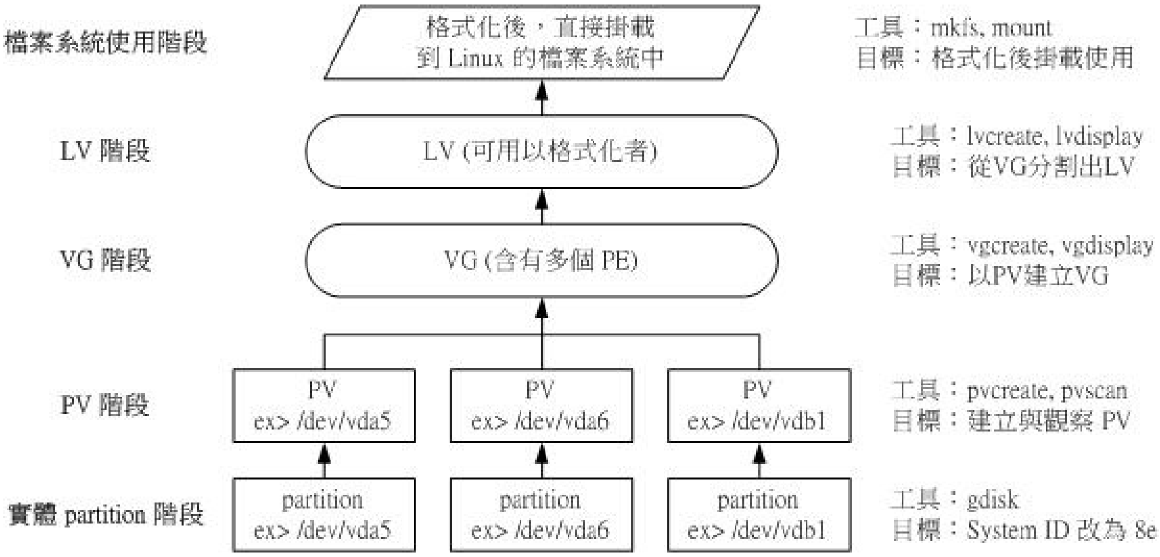


图14.3.1、PE 与 VG 的相关性图示

如上图所示，VG 内的 PE 会分给虚线部分的 LV，如果未来这个 VG 要扩充的话，加上其他的 PV 即可。 而最重要的 LV 如果要扩充的话，也是通过加入 VG 内没有使用到的 PE 来扩充的！

- 实作流程

通过 PV, VG, LV 的规划之后，再利用 mkfs 就可以将你的 LV 格式化成为可以利用的文件系统了！而且这个文件系统的容量在未来还能够进行扩充或减少， 而且里面的数据还不会被影响！实在是很“福气啦！”那实作方面要如何进行呢？很简单呢！ 整个流程由基础到最终的结果可以这样看：



图

14.3.2、LVM 各元件的实现流程图示

如此一来，我们就可以利用 LV 这个玩意儿来进行系统的挂载了。不过，你应该要觉得奇怪的是，那么我的数据写入这个 LV 时，到底他是怎么写入硬盘当中的？ 呵呵！好问题～其实，依据写入机制的不同，而有两种方式：

- 线性模式 （linear）：假如我将 /dev/vda1, /dev/vdb1 这两个 partition 加入到 VG 当中，并且整个 VG 只有一个 LV 时，那么所谓的线性模式就是：当 /dev/vda1 的容量用完之后，/dev/vdb1 的硬盘才会被使用到， 这也是我们所建议的模式。
- 交错模式 （triped）：那什么是交错模式？很简单啊，就是我将一笔数据拆成两部分，分别写入 /dev/vda1 与 /dev/vdb1 的意思，感觉上有点像 RAID 0 啦！如此一来，一份数据用两颗硬盘来写入，理论上，读写的性能会比较好。

基本上，LVM 最主要的用处是在实现一个可以弹性调整容量的文件系统上， 而不是在创建一个性能为主的磁盘上，所以，我们应该利用的是 LVM 可以弹性管理整个 partition 大小的用途上，而不是着眼在性能上的。因此， LVM 默认的读写模式是线性模式啦！ 如果你使用 triped 模式，要注意，当任何一个 partition “归天”时，所有的数据都会“损毁”的！ 所以啦，不是很适合使用这种模式啦！如果要强调性能与备份，那么就直接使用 RAID 即可， 不需要用到 LVM 啊！

14.3.2 LVM 实作流程

LVM 必需要核心有支持且需要安装 lvm2 这个软件，好佳在的是， CentOS 与其他较新的 distributions 已经默认将 lvm 的支持与软件都安装妥当了！所以你不需要担心这方面的问题！用就对了！

假设你刚刚也是通过同样的方法来处理鸟哥的测试机 RAID 实作，那么现在应该有 5 个可用的分区才对！ 不过，建议你还是得要修改一下 system ID 比较好！将 RAID 的 fd 改为 LVM 的 8e 吧！现在，我们实作 LVM 有点像下面的模样：

- 使用 4 个 partition ，每个 partition 的容量均为 1GB 左右，且 system ID 需要为 8e；
- 全部的 partition 整合成为一个 VG，VG 名称设置为 vbirdvg；且 PE 的大小为 16MB；
- 创建一个名为 vbirdlv 的 LV，容量大约 2G 好了！
- 最终这个 LV 格式化为 xfs 的文件系统，且挂载在 /srv/lvm 中

- 0. Disk 阶段 （实际的磁盘）

鸟哥就不仔细的介绍实体分区了，请您自行参考第七章的 gdisk 来达成下面的范例：

```
[root@study ~]# gdisk -l /dev/vda
Number  Start （sector）    End （sector）  Size       Code  Name
   1            2048            6143   2.0 MiB     EF02
   2            6144         2103295   1024.0 MiB  0700
   3         2103296        65026047   30.0 GiB    8E00
   4        65026048        67123199   1024.0 MiB  8300  Linux filesystem
 5        67123200        69220351   1024.0 MiB  8E00  Linux LVM
   6        69220352        71317503   1024.0 MiB  8E00  Linux LVM
   7        71317504        73414655   1024.0 MiB  8E00  Linux LVM
   8        73414656        75511807   1024.0 MiB  8E00  Linux LVM
   9        75511808        77608959   1024.0 MiB  8E00  Linux LVM
# 其实 system ID 不改变也没关系！只是为了让我们管理员清楚知道该 partition 的内容，
# 所以这里建议还是修订成正确的磁盘内容较佳！
```

上面的 /dev/vda{5,6,7,8} 这 4 个分区就是我们的实体分区！也就是下面会实际用到的信息！至于 /dev/vda9 则先保留下来不使用。 注意看，那个 8e 的出现会导致 system 变成“ Linux LVM ”哩！其实没有设置成为 8e 也没关系， 不过某些 LVM 的侦测指令可能会侦测不到该 partition 就是了！接下来，就一个一个的处理各流程吧！

- 1. PV 阶段

要创建 PV 其实很简单，只要直接使用 pvcreate 即可！我们来谈一谈与 PV 有关的指令吧！

- pvcreate ：将实体 partition 创建成为 PV ；
- pvscan ：搜寻目前系统里面任何具有 PV 的磁盘；
- pvdisplay ：显示出目前系统上面的 PV 状态；
- pvremove ：将 PV 属性移除，让该 partition 不具有 PV 属性。

那就直接来瞧一瞧吧！

```
# 1\. 检查有无 PV 在系统上，然后将 /dev/vda{5-8} 创建成为 PV 格式
[root@study ~]# pvscan
  PV /dev/vda3   VG centos   lvm2 [30.00 GiB / 14.00 GiB free]
  Total: 1 [30.00 GiB] / in use: 1 [30.00 GiB] / in no VG: 0 [0   ]
# 其实安装的时候，我们就有使用 LVM 了喔！所以会有 /dev/vda3 存在的！

[root@study ~]# pvcreate /dev/vda{5,6,7,8}
  Physical volume "/dev/vda5" successfully created
  Physical volume "/dev/vda6" successfully created
  Physical volume "/dev/vda7" successfully created
  Physical volume "/dev/vda8" successfully created
# 这个指令可以一口气创建这四个 partition 成为 PV 啦！注意大括号的用途

[root@study ~]# pvscan
  PV /dev/vda3   VG centos   lvm2 [30.00 GiB / 14.00 GiB free]
 PV /dev/vda8               lvm2 [1.00 GiB]
  PV /dev/vda5              lvm2 [1.00 GiB]
  PV /dev/vda7              lvm2 [1.00 GiB]
  PV /dev/vda6              lvm2 [1.00 GiB]
  Total: 5 [34.00 GiB] / in use: 1 [30.00 GiB] / in no VG: 4 [4.00 GiB]
# 这就分别显示每个 PV 的信息与系统所有 PV 的信息。尤其最后一行，显示的是：
# 整体 PV 的量 / 已经被使用到 VG 的 PV 量 / 剩余的 PV 量

# 2\. 更详细的列示出系统上面每个 PV 的个别信息：
[root@study ~]# pvdisplay /dev/vda5
  "/dev/vda5" is a new physical volume of "1.00 GiB"
  --- NEW Physical volume ---
  PV Name               /dev/vda5  &lt;==实际的 partition 设备名称
  VG Name                          &lt;==因为尚未分配出去，所以空白！
  PV Size               1.00 GiB   &lt;==就是容量说明
  Allocatable           NO         &lt;==是否已被分配，结果是 NO
  PE Size               0          &lt;==在此 PV 内的 PE 大小
  Total PE              0          &lt;==共分区出几个 PE
  Free PE               0          &lt;==没被 LV 用掉的 PE
  Allocated PE          0          &lt;==尚可分配出去的 PE 数量
  PV UUID               Cb717z-lShq-6WXf-ewEj-qg0W-MieW-oAZTR6
# 由于 PE 是在创建 VG 时才给予的参数，因此在这里看到的 PV 里头的 PE 都会是 0
# 而且也没有多余的 PE 可供分配 （allocatable）。
```

讲是很难，作是很简单！这样就将 PV 创建了起来啰！简单到不行吧！ ^_^！继续来玩 VG 去！

- 2. VG 阶段

创建 VG 及 VG 相关的指令也不少，我们来看看：

- vgcreate ：就是主要创建 VG 的指令啦！他的参数比较多，等一下介绍。
- vgscan ：搜寻系统上面是否有 VG 存在？
- vgdisplay ：显示目前系统上面的 VG 状态；
- vgextend ：在 VG 内增加额外的 PV ；
- vgreduce ：在 VG 内移除 PV；
- vgchange ：设置 VG 是否启动 （active）；
- vgremove ：删除一个 VG 啊！

与 PV 不同的是， VG 的名称是自订的！我们知道 PV 的名称其实就是 partition 的设备文件名， 但是这个 VG 名称则可以随便你自己取啊！在下面的例子当中，我将 VG 名称取名为 vbirdvg 。创建这个 VG 的流程是这样的：

```
[root@study ~]# vgcreate [-s N[mgt]] VG名称 PV名称
选项与参数：
-s ：后面接 PE 的大小 （size） ，单位可以是 m, g, t （大小写均可）

# 1\. 将 /dev/vda5-7 创建成为一个 VG，且指定 PE 为 16MB 喔！
[root@study ~]# vgcreate -s 16M vbirdvg /dev/vda{5,6,7}
  Volume group "vbirdvg" successfully created

[root@study ~]# vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "vbirdvg" using metadata type lvm2  # 我们手动制作的
  Found volume group "centos" using metadata type lvm2   # 之前系统安装时作的

[root@study ~]# pvscan
 PV /dev/vda5   VG vbirdvg   lvm2 [1008.00 MiB / 1008.00 MiB free]
  PV /dev/vda6   VG vbirdvg   lvm2 [1008.00 MiB / 1008.00 MiB free]
  PV /dev/vda7   VG vbirdvg   lvm2 [1008.00 MiB / 1008.00 MiB free]
  PV /dev/vda3   VG centos    lvm2 [30.00 GiB / 14.00 GiB free]
  PV /dev/vda8                lvm2 [1.00 GiB]
  Total: 5 [33.95 GiB] / in use: 4 [32.95 GiB] / in no VG: 1 [1.00 GiB]
# 嘿嘿！发现没！有三个 PV 被用去，剩下 1 个 /dev/vda8 的 PV 没被用掉！

[root@study ~]# vgdisplay vbirdvg
  --- Volume group ---
  VG Name               vbirdvg
  System ID
  Format                lvm2
  Metadata Areas        3
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                3
  Act PV                3
  VG Size               2.95 GiB        &lt;==整体的 VG 容量有这么大
  PE Size               16.00 MiB       &lt;==内部每个 PE 的大小
 Total PE              189             &lt;==总共的 PE 数量共有这么多！
  Alloc PE / Size       0 / 0
  Free  PE / Size       189 / 2.95 GiB &lt;==尚可配置给 LV 的 PE数量/总容量有这么多！
  VG UUID               Rx7zdR-y2cY-HuIZ-Yd2s-odU8-AkTW-okk4Ea
# 最后那三行指的就是 PE 能够使用的情况！由于尚未切出 LV，因此所有的 PE 均可自由使用。
```

这样就创建一个 VG 了！假设我们要增加这个 VG 的容量，因为我们还有 /dev/vda8 嘛！此时你可以这样做：

```
# 2\. 将剩余的 PV （/dev/vda8） 丢给 vbirdvg 吧！
[root@study ~]# vgextend vbirdvg /dev/vda8
  Volume group "vbirdvg" successfully extended

[root@study ~]# vgdisplay vbirdvg
....（前面省略）....
  VG Size               3.94 GiB
  PE Size               16.00 MiB
 Total PE              252
  Alloc PE / Size       0 / 0
  Free  PE / Size       252 / 3.94 GiB
# 基本上，不难吧！这样就可以抽换整个 VG 的大小啊！
```

我们多了一个设备喔！接下来为这个 vbirdvg 进行分区吧！通过 LV 功能来处理！

- 3. LV 阶段

创造出 VG 这个大磁盘之后，再来就是要创建分区区啦！这个分区区就是所谓的 LV 啰！假设我要将刚刚那个 vbirdvg 磁盘，分区成为 vbirdlv ，整个 VG 的容量都被分配到 vbirdlv 里面去！先来看看能使用的指令后，就直接工作了先！

- lvcreate ：创建 LV 啦！
- lvscan ：查询系统上面的 LV ；
- lvdisplay ：显示系统上面的 LV 状态啊！
- lvextend ：在 LV 里面增加容量！
- lvreduce ：在 LV 里面减少容量；
- lvremove ：删除一个 LV ！
- lvresize ：对 LV 进行容量大小的调整！

```
[root@study ~]# lvcreate [-L N[mgt]] [-n LV名称] VG名称
[root@study ~]# lvcreate [-l N] [-n LV名称] VG名称
选项与参数：
-L  ：后面接容量，容量的单位可以是 M,G,T 等，要注意的是，最小单位为 PE，
      因此这个数量必须要是 PE 的倍数，若不相符，系统会自行计算最相近的容量。
-l  ：后面可以接 PE 的“个数”，而不是数量。若要这么做，得要自行计算 PE 数。
-n  ：后面接的就是 LV 的名称啦！
更多的说明应该可以自行查阅吧！ man lvcreate

# 1\. 将 vbirdvg 分 2GB 给 vbirdlv 喔！
[root@study ~]# lvcreate -L 2G -n vbirdlv vbirdvg
  Logical volume "vbirdlv" created
# 由于本案例中每个 PE 为 16M ，如果要用 PE 的数量来处理的话，那使用下面的指令也 OK喔！
# lvcreate -l 128 -n vbirdlv vbirdvg

[root@study ~]# lvscan
  ACTIVE            '/dev/vbirdvg/vbirdlv' [2.00 GiB] inherit  &lt;==新增加的一个 LV 啰！
  ACTIVE            '/dev/centos/root' [10.00 GiB] inherit
  ACTIVE            '/dev/centos/home' [5.00 GiB] inherit
  ACTIVE            '/dev/centos/swap' [1.00 GiB] inherit

[root@study ~]# lvdisplay /dev/vbirdvg/vbirdlv
  --- Logical volume ---
  LV Path                /dev/vbirdvg/vbirdlv   # 这个是 LV 的全名喔！
  LV Name                vbirdlv
  VG Name                vbirdvg
  LV UUID                QJJrTC-66sm-878Y-o2DC-nN37-2nFR-0BwMmn
  LV Write Access        read/write
  LV Creation host, time study.centos.vbird, 2015-07-28 02:22:49 +0800
  LV Status              available
  # open                 0
  LV Size                2.00 GiB               # 容量就是这么大！
  Current LE             128
  Segments               3
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     8192
  Block device           253:3
```

如此一来，整个 LV partition 也准备好啦！接下来，就是针对这个 LV 来处理啦！要特别注意的是， VG 的名称为 vbirdvg ， 但是 LV 的名称必须使用全名！亦即是 /dev/vbirdvg/vbirdlv 才对喔！ 后续的处理都是这样的！这点初次接触 LVM 的朋友很容易搞错！

- 文件系统阶段

这个部分鸟哥我就不再多加解释了！直接来进行吧！

```
# 1\. 格式化、挂载与观察我们的 LV 吧！
[root@study ~]# mkfs.xfs /dev/vbirdvg/vbirdlv &lt;==注意 LV 全名！
[root@study ~]# mkdir /srv/lvm
[root@study ~]# mount /dev/vbirdvg/vbirdlv /srv/lvm
[root@study ~]# df -Th /srv/lvm
Filesystem                  Type  Size  Used Avail Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv xfs   2.0G   33M  2.0G   2% /srv/lvm

[root@study ~]# cp -a /etc /var/log /srv/lvm
[root@study ~]# df -Th /srv/lvm
Filesystem                  Type  Size  Used Avail Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv xfs   2.0G  152M  1.9G   8% /srv/lvm  &lt;==确定是可用的啊！
```

通过这样的功能，我们现在已经创建好一个 LV 了！你可以自由的应用 /srv/lvm 内的所有资源！

14.3.3 放大 LV 容量

我们不是说 LVM 最大的特色就是弹性调整磁盘容量吗？好！那我们就来处理一下，如果要放大 LV 的容量时， 该如何进行完整的步骤呢？其实一点都不难喔！如果你回去看图 14.3.2 的话，那么你会知道放大文件系统时， 需要下面这些流程的：

1. VG 阶段需要有剩余的容量：因为需要放大文件系统，所以需要放大 LV，但是若没有多的 VG 容量， 那么更上层的 LV 与文件系统就无法放大的。因此，你得要用尽各种方法来产生多的 VG 容量才行。一般来说，如果 VG 容量不足， 最简单的方法就是再加硬盘！然后将该硬盘使用上面讲过的 pvcreate 及 vgextend 增加到该 VG 内即可！

2. LV 阶段产生更多的可用容量：如果 VG 的剩余容量足够了， 此时就可以利用 lvresize 这个指令来将剩余容量加入到所需要增加的 LV 设备内！过程相当简单！

3. 文件系统阶段的放大：我们的 Linux 实际使用的其实不是 LV 啊！而是 LV 这个设备内的文件系统！ 所以一切最终还是要以文件系统为依归！目前在 Linux 环境下，鸟哥测试过可以放大的文件系统有 XFS 以及 EXT 家族！ 至于缩小仅有 EXT 家族，目前 XFS 文件系统并不支持文件系统的容量缩小喔！要注意！要注意！XFS 放大文件系统通过简单的 xfs_growfs 指令即可！

其中最后一个步骤最重要！我们在第七章当中知道， 整个文件系统在最初格式化的时候就创建了 inode/block/superblock 等信息，要改变这些信息是很难的！ 不过因为文件系统格式化的时候创建的是多个 block group ，因此我们可以通过在文件系统当中增加 block group 的方式来增减文件系统的量！而增减 block group 就是利用 xfs_growfs 啰！所以最后一步是针对文件系统来处理的， 前面几步则是针对 LVM 的实际容量大小！



Tips 因此，严格说起来，放大文件系统并不是没有进行“格式化”喔！放大文件系统时，格式化的位置在于该设备后来新增的部份，设备的前面已经存在的文件系统则没有变化。 而新增的格式化过的数据，再反馈回原本的 supberblock 这样而已！

让我们来实作个范例，假设我们想要针对 /srv/lvm 再增加 500M 的容量，该如何处置？

```
# 1\. 由前面的过程我们知道 /srv/lvm 是 /dev/vbirdvg/vbirdlv 这个设备，所以检查 vbirdvg 吧！
[root@study ~]# vgdisplay vbirdvg
  --- Volume group ---
  VG Name               vbirdvg
  System ID
  Format                lvm2
  Metadata Areas        4
  Metadata Sequence No  3
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                1
  Open LV               1
  Max PV                0
  Cur PV                4
  Act PV                4
  VG Size               3.94 GiB
  PE Size               16.00 MiB
  Total PE              252
  Alloc PE / Size       128 / 2.00 GiB
 Free  PE / Size       124 / 1.94 GiB    # 看起来剩余容量确实超过 500M 的！
  VG UUID               Rx7zdR-y2cY-HuIZ-Yd2s-odU8-AkTW-okk4Ea
# 既然 VG 的容量够大了！所以直接来放大 LV 吧！！

# 2\. 放大 LV 吧！利用 lvresize 的功能来增加！
[root@study ~]# lvresize -L +500M /dev/vbirdvg/vbirdlv
  Rounding size to boundary between physical extents: 512.00 MiB
  Size of logical volume vbirdvg/vbirdlv changed from 2.00 GiB （128 extents） to 2.50 GiB
（160 extents）.
  Logical volume vbirdlv successfully resized
# 这样就增加了 LV 了喔！lvresize 的语法很简单，基本上同样通过 -l 或 -L 来增加！
# 若要增加则使用 + ，若要减少则使用 - ！详细的选项请参考 man lvresize 啰！

[root@study ~]# lvscan
 ACTIVE            '/dev/vbirdvg/vbirdlv' [2.50 GiB] inherit
  ACTIVE            '/dev/centos/root' [10.00 GiB] inherit
  ACTIVE            '/dev/centos/home' [5.00 GiB] inherit
  ACTIVE            '/dev/centos/swap' [1.00 GiB] inherit
# 可以发现 /dev/vbirdvg/vbirdlv 容量由 2G 增加到 2.5G 啰！

[root@study ~]# df -Th /srv/lvm
Filesystem                  Type  Size  Used Avail Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv xfs   2.0G  111M  1.9G   6% /srv/lvm
```

看到了吧？最终的结果中 LV 真的有放大到 2.5GB 喔！但是文件系统却没有相对增加！而且，我们的 LVM 可以线上直接处理，并不需要特别给他 umount 哩！真是人性化！ 但是还是得要处理一下文件系统的容量啦！开始观察一下文件系统，然后使用 xfs_growfs 来处理一下吧！

```
# 3.1 先看一下原本的文件系统内的 superblock 记录情况吧！
[root@study ~]# xfs_info /srv/lvm
meta-data=/dev/mapper/vbirdvg-vbirdlv isize=256    agcount=4, agsize=131072 blks
         =                            sectsz=512   attr=2, projid32bit=1
         =                            crc=0        finobt=0
data     =                            bsize=4096   blocks=524288, imaxpct=25
         =                            sunit=0      swidth=0 blks
naming   =version 2                   bsize=4096   ascii-ci=0 ftype=0
log      =internal                    bsize=4096   blocks=2560, version=2
         =                            sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                        extsz=4096   blocks=0, rtextents=0

[root@study ~]# xfs_growfs /srv/lvm  # 这一步骤才是最重要的！
[root@study ~]# xfs_info /srv/lvm
meta-data=/dev/mapper/vbirdvg-vbirdlv isize=256    agcount=5, agsize=131072 blks
         =                            sectsz=512   attr=2, projid32bit=1
         =                            crc=0        finobt=0
data     =                            bsize=4096   blocks=655360, imaxpct=25
         =                            sunit=0      swidth=0 blks
naming   =version 2                   bsize=4096   ascii-ci=0 ftype=0
log      =internal                    bsize=4096   blocks=2560, version=2
         =                            sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                        extsz=4096   blocks=0, rtextents=0

[root@study ~]# df -Th /srv/lvm
Filesystem                  Type  Size  Used Avail Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv xfs   2.5G  111M  2.4G   5% /srv/lvm

[root@study ~]# ls -l /srv/lvm
drwxr-xr-x. 131 root root 8192 Jul 28 00:12 etc
drwxr-xr-x.  16 root root 4096 Jul 28 00:01 log
# 刚刚复制进去的数据可还是存在的喔！并没有消失不见！
```

在上表中，注意看两次 xfs_info 的结果，你会发现到 1）整个 block group （agcount） 的数量增加一个！那个 block group 就是纪录新的设备容量之文件系统所在！ 而你也会 2）发现整体的 block 数量增加了！这样整个文件系统就给他放大了！同时，使用 df 去查阅时，就眞的看到增加的量了吧！ 文件系统的放大可以在 On-line 的环境下实作耶！超棒的！

最后，请注意！目前的 XFS 文件系统中，并没有缩小文件系统容量的设计！也就是说，文件系统只能放大不能缩小喔！如果你想要保有放大、缩小的本事， 那还请回去使用 EXT 家族最新的 EXT4 文件系统啰！XFS 目前是办不到的！

14.3.4 使用 LVM thin Volume 让 LVM 动态自动调整磁盘使用率

想像一个情况，你有个目录未来会使用到大约 5T 的容量，但是目前你的磁盘仅有 3T，问题是，接下来的两个月你的系统都还不会超过 3T 的容量， 不过你想要让用户知道，就是他最多有 5T 可以使用就是了！而且在一个月内你确实可以将系统提升到 5T 以上的容量啊！ 你又不想要在提升容量后才放大到 5T！那可以怎么办？呵呵！这时可以考虑“实际用多少才分配多少容量给 LV 的 LVM Thin Volume ”功能！

另外，再想像一个环境，如果你需要有 3 个 10GB 的磁盘来进行某些测试，问题是你的环境仅有 5GB 的剩余容量，再传统的 LVM 环境下， LV 的容量是一开始就分配好的，因此你当然没有办法在这样的环境中产生出 3 个 10GB 的设备啊！而且更呕的是，那个 10GB 的设备其

实每个实际使用率都没有超过 10%， 也就是总用量目前仅会到 3GB 而已！但...我实际就有 5GB 的容量啊！为何不给我做出 3 个只用 1GB 的 10GB 设备呢？有啊！就还是 LVM thin Volume 啊！

什么是 LVM thin Volume 呢？这东西其实挺好玩的，他的概念是：先创建一个可以实支实付、用多少容量才分配实际写入多少容量的磁盘容量储存池 （thin pool）， 然后再由这个 thin pool 去产生一个“指定要固定容量大小的 LV 设备”，这个 LV 就有趣了！虽然你会看到“宣告上，他的容量可能有 10GB ，但实际上， 该设备用到多少容量时，才会从 thin pool 去实际取得所需要的容量”！就如同上面的环境说的，可能我们的 thin pool 仅有 1GB 的容量， 但是可以分配给一个 10GB 的 LV 设备！而该设备实际使用到 500M 时，整个 thin pool 才分配 500M 给该 LV 的意思！当然啦！ 在所有由 thin pool 所分配出来的 LV 设备中，总实际使用量绝不能超过 thin pool 的最大实际容量啊！如这个案例说的， thin pool 仅有 1GB， 那所有的由这个 thin pool 创建出来的 LV 设备内的实际用量，就绝不能超过 1GB 啊！

我们来实作个环境好了！刚刚鸟哥的 vbirdvg 应该还有剩余容量，那么请这样作看看：

1. 由 vbirdvg 的剩余容量取出 1GB 来做出一个名为 vbirdtpool 的 thin pool LV 设备，这就是所谓的磁盘容量储存池 （thin pool）
2. 由 vbirdvg 内的 vbirdtpool 产生一个名为 vbirdthin1 的 10GB LV 设备
3. 将此设备实际格式化为 xfs 文件系统，并且挂载于 /srv/thin 目录内！

话不多说，我们来实验看看！

```
# 1\. 先以 lvcreate 来创建 vbirdtpool 这个 thin pool 设备：
[root@study ~]# lvcreate -L 1G -T vbirdvg/vbirdtpool  # 最重要的创建指令
[root@study ~]# lvdisplay /dev/vbirdvg/vbirdtpool
  --- Logical volume ---
  LV Name                vbirdtpool
  VG Name                vbirdvg
  LV UUID                p3sLAg-Z8jT-tBuT-wmEL-1wKZ-jrGP-0xmLtk
  LV Write Access        read/write
  LV Creation host, time study.centos.vbird, 2015-07-28 18:27:32 +0800
  LV Pool metadata       vbirdtpool_tmeta
  LV Pool data           vbirdtpool_tdata
  LV Status              available
  # open                 0
 LV Size                1.00 GiB   # 总共可分配出去的容量
  Allocated pool data    0.00%      # 已分配的容量百分比
  Allocated metadata     0.24%      # 已分配的中介数据百分比
  Current LE             64
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     8192
  Block device           253:6
# 非常有趣吧！竟然在 LV 设备中还可以有再分配 （Allocated） 的项目耶！果然是储存池！

[root@study ~]# lvs vbirdvg  # 语法为 lvs VGname
  LV         VG      Attr       LSize Pool Origin Data%  Meta%  Move Log Cpy%Sync Convert
  vbirdlv    vbirdvg -wi-ao---- 2.50g
  vbirdtpool vbirdvg twi-a-tz-- 1.00g             0.00   0.24
# 这个 lvs 指令的输出更加简单明了！直接看比较清晰！

# 2\. 开始创建 vbirdthin1 这个有 10GB 的设备，注意！必须使用 --thin 与 vbirdtpool 链接喔！
[root@study ~]# lvcreate -V 10G -T vbirdvg/vbirdtpool -n vbirdthin1

[root@study ~]# lvs vbirdvg
  LV         VG      Attr       LSize  Pool       Origin Data%  Meta%  Move Log Cpy%Sync
  vbirdlv    vbirdvg -wi-ao----  2.50g
 vbirdthin1 vbirdvg Vwi-a-tz-- 10.00g vbirdtpool        0.00
  vbirdtpool vbirdvg twi-aotz--  1.00g                   0.00   0.27
# 很有趣吧！明明连 vbirdvg 这个 VG 都没有足够大到 10GB 的容量，通过 thin pool
# 竟然就产生了 10GB 的 vbirdthin1 这个设备了！好有趣！

# 3\. 开始创建文件系统
[root@study ~]# mkfs.xfs /dev/vbirdvg/vbirdthin1
[root@study ~]# mkdir /srv/thin
[root@study ~]# mount /dev/vbirdvg/vbirdthin1 /srv/thin
[root@study ~]# df -Th /srv/thin
Filesystem                     Type  Size  Used Avail Use% Mounted on
/dev/mapper/vbirdvg-vbirdthin1 xfs    10G   33M   10G   1% /srv/thin
# 真的有 10GB 耶！！

# 4\. 测试一下容量的使用！创建 500MB 的文件，但不可超过 1GB 的测试为宜！
[root@study ~]# dd if=/dev/zero of=/srv/thin/test.img bs=1M count=500
[root@study ~]# lvs vbirdvg
  LV         VG      Attr       LSize  Pool       Origin Data%  Meta%  Move Log Cpy%Sync
  vbirdlv    vbirdvg -wi-ao----  2.50g
  vbirdthin1 vbirdvg Vwi-aotz-- 10.00g vbirdtpool        4.99
  vbirdtpool vbirdvg twi-aotz--  1.00g                   49.93  1.81
# 很要命！这时已经分配出 49% 以上的容量了！而 vbirdthin1 却只看到用掉 5% 而已！
# 所以鸟哥认为，这个 thin pool 非常好用！但是在管理上，得要特别特别的留意！
```

这就是用多少算多少的 thin pool 实作方式！基本上，用来骗人挺吓人的！小小的一个磁盘可以仿真出好多容量！但实际上，真的可用容量就是实际的磁盘储存池内的容量！ 如果突破该容量，这个 thin pool 可是会爆炸而让数据损毁的！要注意！要注意！

14.3.5 LVM 的 LV 磁盘快照

现在你知道 LVM 的好处咯，未来如果你有想要增加某个 LVM 的容量时，就可以通过这个放大的功能来处理。 那么 LVM 除了这些功能之外，还有什么能力呢？其实他还有一个重要的能力，那就是 LV 磁盘的快照 （snapshot）。 什么是 LV 磁盘快照啊？快照就是将当时的系统信息记录下来，就好像照相记录一般！ 未来若有任何数据更动了，则原始数据会被搬移到快照区，没有被更动的区域则由快照区与文件系统共享。 用讲的好像很难懂，我们用图解说明一下好了：

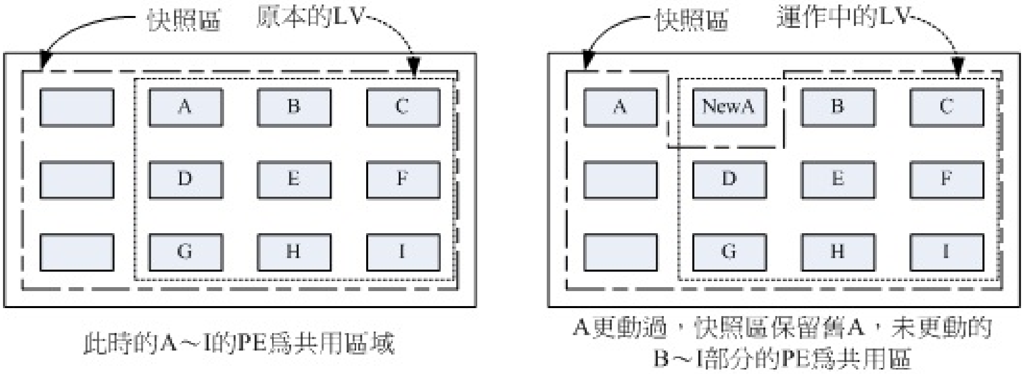


图14.3.3、LVM 快照区域的备份示意图

左图为最初创建 LV 磁盘快照区的状况，LVM 会预留一个区域 （左图的左侧三个 PE 区块） 作为数据存放处。 此时快照区内并没有任何数据，而快照区与系统区共享所有的 PE 数据， 因此你会看到快照区的内容与文件系统是一模一样的。 等到系统运行一阵子后，假设 A 区域的数据被更动了 （上面右图所示） ，则更动前系统会将该区域的数据移动到快照区， 所以在右图的快照区被占用了一块 PE 成为 A，而其他 B 到 I 的区块则还是与文件系统共享！

照这样的情况来看，LVM 的磁盘快照是非常棒的“备份工具”，因为他只有备份有被更动到的数据， 文件系统内没有被变更的数据依旧保持在原本的区块内，但是 LVM 快照功能会知道那些数据放置在哪里， 因此“快照”当时的文件系统就得以“备份”下来，且快照所占用的容量又非常小！所以您说，这不是很棒的工具又是什么？

那么快照区要如何创建与使用呢？首先，由于快照区与原本的 LV 共享很多 PE 区块，因此快照区与被快照的 LV 必须要在同一个 VG 上头。

另外，或许你跟鸟哥一样，会想到说：“咦！ 我们能不能使用 thin pool 的功能来制作快照”呢？老实说，是可以的！不过使用上面的限制非常的多！包括最好要在同一个 thin pool 内的原始 LV 磁盘， 如果为非 thin pool 内的原始 LV 磁盘快照，则该磁盘快照“不可以写入”，亦即 LV 磁盘要设置成只读才行！同时， 使用 thin pool 做出来的快照，通常都是不可启动 （inactive） 的默认情况，启动又有点麻烦～所以，至少目前 （CentOS 7.x） 的环境下， 鸟哥还不是很建议你使用 thin pool 快照喔！

下面我们针对传统 LV 磁盘进行快照的创建，大致流程为：

- 预计被拿来备份的原始 LV 为 /dev/vbirdvg/vbirdlv 这个东西～

- 使用传统方式快照创建，原始碟为 /dev/vbirdvg/vbirdlv，快照名称为 vbirdsnap1，容量为 vbirdvg 的所有剩余容量
- 传统快照区的创建

```
# 1\. 先观察 VG 还剩下多少剩余容量
[root@study ~]# vgdisplay vbirdvg
....（其他省略）....
  Total PE              252
  Alloc PE / Size       226 / 3.53 GiB
  Free  PE / Size       26 / 416.00 MiB
# 就只有剩下 26 个 PE 了！全部分配给 vbirdsnap1 啰！

# 2\. 利用 lvcreate 创建 vbirdlv 的快照区，快照被取名为 vbirdsnap1，且给予 26 个 PE
[root@study ~]# lvcreate -s -l 26 -n vbirdsnap1 /dev/vbirdvg/vbirdlv
  Logical volume "vbirdsnap1" created
# 上述的指令中最重要的是那个 -s 的选项！代表是 snapshot 快照功能之意！
# -n 后面接快照区的设备名称， /dev/.... 则是要被快照的 LV 完整文件名。
# -l 后面则是接使用多少个 PE 来作为这个快照区使用。

[root@study ~]# lvdisplay /dev/vbirdvg/vbirdsnap1
  --- Logical volume ---
  LV Path                /dev/vbirdvg/vbirdsnap1
  LV Name                vbirdsnap1
  VG Name                vbirdvg
  LV UUID                I3m3Oc-RIvC-unag-DiiA-iQgI-I3z9-0OaOzR
  LV Write Access        read/write
  LV Creation host, time study.centos.vbird, 2015-07-28 19:21:44 +0800
  LV snapshot status     active destination for vbirdlv
  LV Status              available
  # open                 0
  LV Size                2.50 GiB    # 原始碟，就是 vbirdlv 的原始容量
  Current LE             160
  COW-table size         416.00 MiB  # 这个快照能够纪录的最大容量！
  COW-table LE           26
  Allocated to snapshot  0.00%       # 目前已经被用掉的容量！
  Snapshot chunk size    4.00 KiB
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     8192
  Block device           253:11
```

您看看！这个 /dev/vbirdvg/vbirdsnap1 快照区就被创建起来了！而且他的 VG 量竟然与原本的 /dev/vbirdvg/vbirdlv 相同！也就是说，如果你真的挂载这个设备时，看到的数据会跟原本的 vbirdlv 相同喔！我们就来测试看看：

```
[root@study ~]# mkdir /srv/snapshot1
[root@study ~]# mount -o nouuid /dev/vbirdvg/vbirdsnap1 /srv/snapshot1
[root@study ~]# df -Th /srv/lvm /srv/snapshot1
Filesystem                     Type  Size  Used Avail Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv    xfs   2.5G  111M  2.4G   5% /srv/lvm
/dev/mapper/vbirdvg-vbirdsnap1 xfs   2.5G  111M  2.4G   5% /srv/snapshot1
# 有没有看到！这两个咚咚竟然是一模一样喔！我们根本没有动过
# /dev/vbirdvg/vbirdsnap1 对吧！不过这里面会主动记录原 vbirdlv 的内容！
```

因为 XFS 不允许相同的 UUID 文件系统的挂载，因此我们得要加上那个 nouuid 的参数，让文件系统忽略相同的 UUID 所造成的问题！ 没办法啊！因为快照出来的文件系统当然是会一模一样的！

- 利用快照区复原系统

首先，我们来玩一下，如何利用快照区复原系统吧！不过你要注意的是，你要复原的数据量不能够高于快照区所能负载的实际容量。由于原始数据会被搬移到快照区， 如果你的快照区不够大，若原始数据被更动的实际数据量比快照区大，那么快照区当然容纳不了，这时候快照功能会失效喔！

我们的 /srv/lvm 已经有 /srv/lvm/etc, /srv/lvm/log 等目录了，接下来我们将这个文件系统的内容作个变更， 然后再以快照区数据还原看看：

```
# 1\. 先将原本的 /dev/vbirdvg/vbirdlv 内容作些变更，增增减减一些目录吧！
[root@study ~]# df -Th /srv/lvm /srv/snapshot1
Filesystem                     Type  Size  Used Avail Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv    xfs   2.5G  111M  2.4G   5% /srv/lvm
/dev/mapper/vbirdvg-vbirdsnap1 xfs   2.5G  111M  2.4G   5% /srv/snapshot1

[root@study ~]# cp -a /usr/share/doc /srv/lvm
[root@study ~]# rm -rf /srv/lvm/log
[root@study ~]# rm -rf /srv/lvm/etc/sysconfig
[root@study ~]# df -Th /srv/lvm /srv/snapshot1
Filesystem                     Type  Size  Used Avail Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv    xfs   2.5G  146M  2.4G   6% /srv/lvm
/dev/mapper/vbirdvg-vbirdsnap1 xfs   2.5G  111M  2.4G   5% /srv/snapshot1
[root@study ~]# ll /srv/lvm /srv/snapshot1
/srv/lvm:
total 60
drwxr-xr-x. 887 root root 28672 Jul 20 23:03 doc
drwxr-xr-x. 131 root root  8192 Jul 28 00:12 etc

/srv/snapshot1:
total 16
drwxr-xr-x. 131 root root 8192 Jul 28 00:12 etc
drwxr-xr-x.  16 root root 4096 Jul 28 00:01 log
# 两个目录的内容看起来已经不太一样了喔！检测一下快照 LV 吧！

[root@study ~]# lvdisplay /dev/vbirdvg/vbirdsnap1
  --- Logical volume ---
  LV Path                /dev/vbirdvg/vbirdsnap1
....（中间省略）....
 Allocated to snapshot  21.47%
# 鸟哥仅列出最重要的部份！就是全部的容量已经被用掉了 21.4% 啰！

# 2\. 利用快照区将原本的 filesystem 备份，我们使用 xfsdump 来处理！
[root@study ~]# xfsdump -l 0 -L lvm1 -M lvm1 -f /home/lvm.dump /srv/snapshot1
# 此时你就会有一个备份数据，亦即是 /home/lvm.dump 了！
```

为什么要备份呢？为什么不可以直接格式化 /dev/vbirdvg/vbirdlv 然后将 /dev/vbirdvg/vbirdsnap1 直接复制给 vbirdlv 呢？ 要知道 vbirdsnap1 其实是 vbirdlv 的快照，因此如果你格式化整个 vbirdlv 时，原本的文件系统所有数据都会被搬移到 vbirdsnap1。 那如果 vbirdsnap1 的容量不够大 （通常也真的不够大），那么部分数据将无法复制到 vbirdsnap1 内，数据当然无法全部还原啊！ 所以才要在上面表格中制作出一个备份文件的！了解乎？

而快照还有另外一个功能，就是你可以比对 /srv/lvm 与 /srv/snapshot1 的内容，就能够发现到最近你到底改了啥咚咚！ 这样也是很不赖啊！您说是吧！ ^_^！接下来让我们准备还原 vbirdlv 的内容吧！

```
# 3\. 将 vbirdsnap1 卸载并移除 （因为里面的内容已经备份起来了）
[root@study ~]# umount /srv/snapshot1
[root@study ~]# lvremove /dev/vbirdvg/vbirdsnap1
Do you really want to remove active logical volume "vbirdsnap1"? [y/n]: y
  Logical volume "vbirdsnap1" successfully removed

[root@study ~]# umount /srv/lvm
[root@study ~]# mkfs.xfs -f /dev/vbirdvg/vbirdlv
[root@study ~]# mount /dev/vbirdvg/vbirdlv /srv/lvm
[root@study ~]# xfsrestore -f /home/lvm.dump -L lvm1 /srv/lvm
[root@study ~]# ll /srv/lvm
drwxr-xr-x. 131 root root 8192 Jul 28 00:12 etc
drwxr-xr-x.  16 root root 4096 Jul 28 00:01 log
# 是否与最初的内容相同啊！这就是通过快照来还原的一个简单的方法啰！
```

- 利用快照区进行各项练习与测试的任务，再以原系统还原快照

换个角度来想想，我们将原本的 vbirdlv 当作备份数据，然后将 vbirdsnap1 当作实际在运行中的数据， 任何测试的动作都在 vbirdsnap1 这个快照区当中测试，那么当测试完毕要将测试的数据删除时，只要将快照区删去即可！ 而要复制一个 vbirdlv 的系统，再作另外一个快照区即可！这样是否非常方便啊？ 这对于教学环境中每年都要帮学生制作一个练习环境主机的测试，非常有帮助呢！



Tips 以前鸟哥老是觉得使用 LVM 的快照来进行备份不太合理，因为还要制作一个备份文件！后来仔细研究并参考徐秉义老师的教材[4]后，才发现 LVM 的快照实在是一个棒到不行的工具！尤其是在虚拟机当中创建多份给同学使用的测试环境， 你只要有一个基础的环境保持住，其他的环境使用快照来提供即可。实时同学将系统搞烂了，你只要将快照区删除， 再重建一个快照区！这样环境就恢复了！天呐！实在是太棒了！ ^_^

14.3.6 LVM 相关指令汇整与 LVM 的关闭

好了，我们将上述用过的一些指令给他汇整一下，提供给您参考参考：

任务	**PV** 阶段	**VG** 阶段	**LV** 阶段	**filesystem（XFS / EXT4）**	
搜寻（scan）	pvscan	vgscan	lvscan	lsblk, blkid	
创建（create）	pvcreate	vgcreate	lvcreate	mkfs.xfs	mkfs.ex
列出（display）	pvdisplay	vgdisplay	lvdisplay	df, mount	
增加（extend）	vgextend	lvextend（lvresize）	xfs_growfs	resize2fs	
减少（reduce）	vgreduce	lvreduce（lvresize）	不支持	resize2fs	
删除（remove）	pvremove	vgremove	lvremove	umount, 重新格式化	
改变容量（resize）	lvresize	xfs_growfs	resize2fs		
改变属性（attribute）	pvchange	vgchange	lvchange	/etc/fstab, remount	

至于文件系统阶段 （filesystem 的格式化处理） 部分，还需要以 xfs*growfs* 来修订文件系统实际的大小才行啊！ ^^ 。至于虽然 LVM 可以弹性的管理你的磁盘容量，但是要注意，如果你想要使用 LVM 管理您的硬盘时，那么在安装的时候就得要做好 LVM 的规划了， 否则未来还是需要先以传统的磁盘增加方式来增加后，移动数据后，才能够进行 LVM 的使用啊！

会玩 LVM 还不行！你必须要会移除系统内的 LVM 喔！因为你的实体 partition 已经被使用到 LVM 去， 如果你还没有将 LVM 关闭就直接将那些 partition 删除或转为其他用途的话，系统是会发生很大的问题的！ 所以啰，你必须要知道如何将 LVM 的设备关闭并移除才行！会不会很难呢？其实不会啦！ 依据以下的流程来处理即可：

1. 先卸载系统上面的 LVM 文件系统 （包括快照与所有 LV）；
2. 使用 lvremove 移除 LV ；
3. 使用 vgchange -a n VGname 让 VGname 这个 VG 不具有 Active 的标志；
4. 使用 vgremove 移除 VG：
5. 使用 pvremove 移除 PV；
6. 最后，使用 fdisk 修改 ID 回来啊！

好吧！那就实际的将我们之前创建的所有 LVM 数据给删除吧！

```
[root@study ~]# umount /srv/lvm /srv/thin /srv/snapshot1
[root@study ~]# lvs vbirdvg
  LV          VG      Attr       LSize  Pool       Origin Data%  Meta%  Move Log Cpy%Sync
  vbirdlv     vbirdvg -wi-a-----  2.50g
  vbirdthin1 vbirdvg Vwi-a-tz-- 10.00g vbirdtpool        4.99
  vbirdtpool vbirdvg twi-aotz--  1.00g                   49.93  1.81
# 要注意！先删除 vbirdthin1 --&gt; vbirdtpool --&gt; vbirdlv 比较好！

[root@study ~]# lvremove /dev/vbirdvg/vbirdthin1 /dev/vbirdvg/vbirdtpool
[root@study ~]# lvremove /dev/vbirdvg/vbirdlv
[root@study ~]# vgchange -a n vbirdvg
  0 logical volume(s) in volume group "vbirdvg" now active

[root@study ~]# vgremove vbirdvg
  Volume group "vbirdvg" successfully removed

[root@study ~]# pvremove /dev/vda{5,6,7,8}
```

最后再用 gdisk 将磁盘的 ID 给他改回来 83 就好啦！整个过程就这样的啦！ ^_^

14.4 重点回顾

- Quota 可公平的分配系统上面的磁盘容量给使用者；分配的资源可以是磁盘容量（block）或可创建文件数量（inode）；
- Quota 的限制可以有 soft/hard/grace time 等重要项目；
- Quota 是针对整个 filesystem 进行限制，XFS 文件系统可以限制目录！
- Quota 的使用必须要核心与文件系统均支持。文件系统的参数必须含有 usrquota, grpquota, prjquota
- Quota 的 xfs_quota 实作的指令有 report, print, limit, timer... 等指令；
- 磁盘阵列（RAID）有硬件与软件之分，Linux 操作系统可支持软件磁盘阵列，通过 mdadm 套件来达成；
- 磁盘阵列创建的考虑依据为“容量”、“性能”、“数据可靠性”等；
- 磁盘阵列所创建的等级常见有的 raid0, raid1, raid1+0, raid5 及 raid6
- 硬件磁盘阵列的设备文件名与 SCSI 相同，至于 software RAID 则为 /dev/md[0-9]
- 软件磁盘阵列的状态可借由 /proc/mdstat 文件来了解；
- LVM 强调的是“弹性的变化文件系统的容量”；
- 与 LVM 有关的元件有： PV/VG/PE/LV 等元件，可以被格式化者为 LV
- 新的 LVM 拥有 LVM thin volume 的功能，能够动态调整磁盘的使用率！
- LVM 拥有快照功能，快照可以记录 LV 的数据内容，并与原有的 LV 共享未更动的数据，备份与还原就变的很简单；
- XFS 通过 xfs_growfs 指令，可以弹性的调整文件系统的大小

14.5 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ）

- 情境仿眞题一：由于 LVM 可以弹性调整 filesystem 的大小，但是缺点是可能没有加速与硬件备份（与快照不同）的功能。 而磁盘阵列则具有性能与备份的功能，但是无法提供类似 LVM 的优点。在此情境中，我们想利用“在 RAID 上面创建 LVM”的功能，以达到两者兼顾的能力。
    - 目标：测试在 RAID 磁盘上面架构 LVM 系统；
    - 需求：需要具有磁盘管理的能力，包括 RAID 与 LVM；
    - 前提：会用到本章创建出来的 /dev/vda5, /dev/vda6, /dev/vda7 三个分区！ 那要如何处理呢？如下的流程一个步骤一个步骤的实施看看吧：
    - 重新处理系统，我们在这个练习当中，需要 /dev/vda5, /dev/vda6, /dev/vda7 创建成一个 RAID5 的 /dev/md0 磁盘！详细的作法这里就不谈了！ 你得要使用 gdisk 来处理成为如下的模样：

    ```
    [root@study ~]# gdisk -l /dev/vda
    Number  Start （sector）    End （sector）  Size       Code  Name
       1            2048            6143   2.0 MiB     EF02
       2            6144         2103295   1024.0 MiB  0700
       3         2103296        65026047   30.0 GiB    8E00
       4        65026048        67123199   1024.0 MiB  8300  Linux filesystem
     5        67123200        69220351   1024.0 MiB  FD00  Linux RAID
       6        69220352        71317503   1024.0 MiB  FD00  Linux RAID
       7        71317504        73414655   1024.0 MiB  FD00  Linux RAID
    ```

    - 开始使用 mdadm 来创建一个简单的 RAID5 阵列！简易的流程如下：

    ```
    [root@study ~]# mdadm --create /dev/md0 --auto=yes  --level=5  \
    > --raid-devices=3 /dev/vda{5,6,7}
    [root@study ~]# mdadm --detail /dev/md0 &#124; grep -i uuid
               UUID : efc7add0:d12ee9ca:e5cb0baa:fbdae4e6
    [root@study ~]# vim /etc/mdadm.conf
    ARRAY /dev/md0 UUID=efc7add0:d12ee9ca:e5cb0baa:fbdae4e6
    ```

    若无出现任何错误讯息，此时你已经具有 /dev/md0 这个磁盘阵列设备了！接下来让我们处理 LVM 吧！

    - 开始处理 LVM ，现在我们假设所有的参数都使用默认值，包括 PE ，然后 VG 名为 raidvg ，LV 名为 raidlv ，下面为基本的流程：

    ```
    [root@study ~]# pvcreate /dev/md0                  <==创建 PV
    [root@study ~]# vgcreate raidvg /dev/md0           <==创建 VG
    [root@study ~]# lvcreate -L 1.5G -n raidlv raidvg  <==创建 LM
    [root@study ~]# lvscan
      ACTIVE            '/dev/raidvg/raidlv' [1.50 GiB] inherit
    ```

这样就搞定了 LVM 了！而且这个 LVM 是架构在 /dev/md0 上面的喔！然后就是文件系统的创建与挂载了！

- 尝试创建成为 XFS 文件系统，且挂载到 /srv/raidlvm 目录下：

```
[root@study ~]# mkfs.xfs /dev/raidvg/raidlv
[root@study ~]# blkid /dev/raidvg/raidlv
/dev/raidvg/raidlv: UUID="4f6a587d-3257-4049-afca-7da1d405117d" TYPE="xfs"
[root@study ~]# vim /etc/fstab
UUID="4f6a587d-3257-4049-afca-7da1d405117d" /srv/raidlvm xfs    defaults 0 0

[root@study ~]# mkdir /srv/raidlvm
[root@study ~]# mount -a
[root@study ~]# df -Th /srv/raidlvm
Filesystem                Type  Size  Used Avail Use% Mounted on
/dev/mapper/raidvg-raidlv xfs   1.5G   33M  1.5G   3% /srv/raidlvm
```

- 上述就是 LVM 架构在 RAID 上面的技巧，之后的动作都能够使用本章的其他管理方式来管理， 包括 RAID 热拔插机制、LVM 放大缩小机制等等。

简答题部分：

- 在前一章的第一个大量新增帐号范例中， 如果我想要让每个用户均具有 soft/hard 各为 40MB/50MB 的容量时，应该如何修改这个 script ？你得先要依据本章的作法，先将 /home 制作好 quota 的环境然后， 你可以在 do...done 内的最后一行，新增一行内容为： xfs_quota -x -c "limit -u bsoft=40M bhard=50M ${username}" /home 这样就可以在制作用户时，指定更新密码且给予 quota 的限制！
- 如果我想要让 RAID 具有保护数据的功能，防止因为硬件损毁而导致数据的遗失，那我应该要选择的 RAID 等级可能有哪些？ （请以本章谈到的等级来思考即可）具有备份数据的有： RAID-1, RAID-5, RAID-6
- 在默认的 LVM 设置中，请问 LVM 能否具有“备份”的功能？是有的，就是那个快照（snopshot） 的功能，此功能即可进行数据的备份！
- 如果你的计算机主机有提供 RAID 0 的功能，你将你的三颗硬盘全部在 BIOS 阶段使用 RAID 芯片整合成为一颗大磁盘， 则此磁盘在 Linux 系统当中的文件名为何？由于硬件磁盘阵列是在 BIOS 阶段完成的，因此 Linux 系统会捉到一个完整的大的 RAID 磁盘，此磁盘的文件名就会是“ /dev/sda ”！ 但如果是 Intel 的芯片组，则还是可能会成为 /dev/md127 等相关的文件名！

14.6 参考资料与延伸阅读

- [1]相关的 XFS 文件系统的 quota 说明，可以参考下面的文件：
  - XFS 官网说明：http://xfs.org/docs/xfsdocs-xml-dev/XFS_User_Guide/tmp/en-US/html/xfs-quotas.html
- [2]若想对 RAID 有更深入的认识，可以参考下面的链接与书目：http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html 杨振和、“操作系统导论：第十一章”、学贯出版社，2006
- [3]详细的 mdstat 说明也可以参考如下网页：https://raid.wiki.kernel.org/index.php/Mdstat
- [4]徐秉义老师在网管人杂志的文章，文章篇名分别是：
  - 磁盘管理：SoftRAID 与 LVM 综合实做应用 （上）
  - 磁盘管理：SoftRAID 与 LVM 综合实做应用 （下）目前文章已经找不到了～可能需要 google 一下旧文章的备份才能看到了！

2002/07/14：第一次完成 2003/02/10：重新编排与加入 FAQ 2003/09/02：加入 quotacheck 发生错误时的解决方法。 2005/09/06：将旧的文章移动到 此处 。 2005/09/06：进行版面风格的转换，并且进行数据的查询，加入 repquota 的简单说明而已！ 2009/03/04：将原本旧的基于 FC4 的文件移动到 此处 。 2009/03/06：加入 warnquota 这玩意儿！挺有趣的哩！ 2009/03/12：加入了 software RAID 与 LVM 的加强说明，尤其是 LVM 的快照 （snapshot） 的说明！ 2009/09/10：修改一些字样之外，增加情境仿真，以及后续的简答题部分题目。 2012/06/14：在解释 PE 的部分有错误！是 Physical Extent 而不是 Physical Extend ！真抱歉！

第十五章、例行性工作调度（crontab）

最近更新日期：20//

学习了基础篇也一阵子了，你会发现到为什么系统常常会主动的进行一些任务？这些任务到底是谁在设置工作的？ 如果你想要让自己设计的备份程序可以自动的在系统下面执行，而不需要手动来启动他，又该如何处置？ 这些例行的工作可能又分为“单一”工作与“循环”工作，在系统内又是哪些服务在负责？ 还有还有，如果你想要每年在老婆的生日前一天就发出一封信件提醒自己不要忘记，可以办的到吗？ 嘿嘿！这些种种要如何处理，就看看这一章先！

15.1 什么是例行性工作调度

每个人或多或少都有一些约会或者是工作，有的工作是例行性的， 例如每年一次的加薪、每个月一次的工作报告、每周一次的午餐会报、每天需要的打卡等等；有的工作则是临时发生的，例如刚好总公司有高官来访，需要你准备演讲器材等等！ 用在生活上面，例如每年的爱人的生日、每天的起床时间等等、还有突发性的 3C 用品大降价 （啊！真希望天天都有！）等等啰。

像上面这些例行性工作，通常你得要记录在行事历上面才能避免忘记！不过，由于我们常常在计算机前面的缘故， 如果计算机系统能够主动的通知我们的话，那么不就轻松多了！嘿嘿！这个时候 Linux 的例行性工作调度就可以派上场了！ 在不考虑硬件与我们服务器的链接状态下，我们的 Linux 可以帮你提醒很多任务，例如：每一天早上 8:00 钟要服务器连接上音响，并启动音乐来唤你起床；而中午 12:00 希望 Linux 可以发一封信到你的邮件信箱，提醒你可以去吃午餐了； 另外，在每年的你爱人生日的前一天，先发封信提醒你，以免忘记这么重要的一天。

那么 Linux 的例行性工作是如何进行调度的呢？所谓的调度就是将这些工作安排执行的流程之意！ 咱们的 Linux 调度就是通过 crontab 与 at 这两个东西！这两个玩意儿有啥异同？就让我们来瞧瞧先！

15.1.1 Linux 工作调度的种类： at, cron

从上面的说明当中，我们可以很清楚的发现两种工作调度的方式：

- 一种是例行性的，就是每隔一定的周期要来办的事项；
- 一种是突发性的，就是这次做完以后就没有的那一种 （ 3C 大降价...）

那么在 Linux 下面如何达到这两个功能呢？那就得使用 at 与 crontab 这两个好东西啰！

- at ：at 是个可以处理仅执行一次就结束调度的指令，不过要执行 at 时， 必须要有 atd 这个服务 （第十七章） 的支持才行。在某些新版的 distributions 中，atd 可能默认并没有启动，那么 at 这个指令就会失效呢！不过我们的 CentOS 默认是启动的！

- crontab ：crontab 这个指令所设置的工作将会循环的一直进行下去！ 可循环的时间为分钟、小时、每周、每月或每年等。crontab 除了可以使用指令执行外，亦可编辑 /etc/crontab 来支持。 至于让 crontab 可以生效的服务则是 crond 这个服务喔！

下面我们先来谈一谈 Linux 的系统到底在做什么事情，怎么有若干多的工作调度在进行呢？然后再回来谈一谈 at 与 crontab 这两个好东西！

15.1.2 CentOS Linux 系统上常见的例行性工作

如果你曾经使用过 Linux 一阵子了，那么你大概会发现到 Linux 会主动的帮我们进行一些工作呢！ 比方说自动的进行线上更新 （on-line update）、自动的进行 updatedb （第六章谈到的 locate 指令） 更新文件名数据库、自动的作登录文件分析 （所以 root 常常会收到标题为 logwatch 的信件） 等等。这是由于系统要正常运行的话， 某些在背景下面的工作必须要定时进行的缘故。基本上 Linux 系统常见的例行性任务有：

- 进行登录文件的轮替 （log rotate） ： Linux 会主动的将系统所发生的各种信息都记录下来，这就是登录文件 （第十八章）。 由于系统会一直记录登录信息，所以登录文件将会越来越大！我们知道大型文件不但占容量还会造成读写性能的困扰， 因此适时的将登录文件数据挪一挪，让旧的数据与新的数据分别存放，则比较可以有效的记录登录信息。这就是 log rotate 的任务！这也是系统必要的例行任务；
- 登录文件分析 logwatch 的任务： 如果系统发生了软件问题、硬件错误、资安问题等，绝大部分的错误信息都会被记录到登录文件中， 因此系统管理员的重要任务之一就是分析登录文件。但你不可能手动通过 vim 等软件去检视登录文件，因为数据太复杂了！ 我们的 CentOS 提供了一只程序“ logwatch ”来主动分析登录信息，所以你会发现，你的 root 老是会收到标题为 logwatch 的信件，那是正常的！你最好也能够看看该信件的内容喔！
- 创建 locate 的数据库： 在第六章我们谈到的 locate 指令时， 我们知道该指令是通过已经存在的文件名数据库来进行系统上文件名的查询。我们的文件名数据库是放置到 /var/lib/mlocate/ 中。 问题是，这个数据库怎么会自动更新啊？嘿嘿！这就是系统的例行性工作所产生的效果啦！系统会主动的进行 updatedb 喔！
- man page 查询数据库的创建： 与 locate 数据库类似的，可提供快速查询的 man page db 也是个数据库，但如果要使用 man page 数据库时，就得要执行 mandb 才能够创建好啊！ 而这个 man page 数据库也是通过系统的例行性工作调度来自动执行的哩！
- RPM 软件登录文件的创建： RPM （第二十二章） 是一种软件管理的机制。由于系统可能会常常变更软件， 包括软件的新安装、非经常性更新等，都会造成软件文件名的差异。为了方便未来追踪，系统也帮我们将文件名作个排序的记录呢！ 有时候系统也会通过调度来帮忙 RPM 数据库的重新创建喔！
- 移除暂存盘： 某些软件在运行中会产生一些暂存盘，但是当这个软件关闭时，这些暂存盘可能并不会主动的被移除。 有些暂存盘则有时间性，如果超过一段时间后，这个暂存盘就没有效用了，此时移除这些暂存盘就是一件重要的工作！ 否则磁盘容量会被耗光。系统通过例行性工作调度执行名为 tmpwatch 的指令来删除这些暂存盘呢！
- 与网络服务有关的分析行为： 如果你有安装类似 WWW 服务器软件 （一个名为 apache 的软件），那么你的 Linux 系统通常就会主动的分析该软件的登录文件。 同时某些凭证与认证的网络信息是否过期的问题，我们的 Linux 系统也会很友好的帮你进行自动检查！

其实你的系统会进行的例行性工作与你安装的软件多寡有关，如果你安装过多的软件，某些服务功能的软件都会附上分析工具， 那么你的系统就会多出一些例行性工作啰！像鸟哥的主机还多加了很多自己撰写的分析工具，以及其他第三方协力软件的分析软件， 嘿嘿！俺的 Linux 工作量可是非常大的哩！因为有这么多的工作需要进行，所以我们当然得要了解例行性工作的处理方式啰！

15.2 仅执行一次的工作调度

首先，我们先来谈谈单一工作调度的运行，那就是 at 这个指令的运行！

15.2.1 atd 的启动与 at 运行的方式

要使用单一工作调度时，我们的 Linux 系统上面必须要有负责这个调度的服务，那就是 atd 这个玩意儿。 不过并非所有的 Linux distributions 都默认会把他打开的，所以呢，某些时刻我们必须要手动将他启用才行。 启用的方法很简单，就是这样：

```
[root@study ~]# systemctl restart atd  # 重新启动 atd 这个服务
[root@study ~]# systemctl enable atd   # 让这个服务开机就自动启动
[root@study ~]# systemctl status atd   # 查阅一下 atd 目前的状态
atd.service - Job spooling tools
   Loaded: loaded (/usr/lib/systemd/system/atd.service; enabled)       # 是否开机启动
   Active: active (running) since Thu 2015-07-30 19:21:21 CST; 23s ago # 是否正在运行中
 Main PID: 26503 (atd)
   CGroup: /system.slice/atd.service
           └─26503 /usr/sbin/atd -f

Jul 30 19:21:21 study.centos.vbird systemd[1]: Starting Job spooling tools...
Jul 30 19:21:21 study.centos.vbird systemd[1]: Started Job spooling tools.
```

重点就是要看到上表中的特殊字体，包括“ enabled ”以及“ running ”时，这才是 atd 真的有在运行的意思喔！这部份我们在第十七章会谈及。

- at 的运行方式

既然是工作调度，那么应该会有产生工作的方式，并且将这些工作排进行程表中啰！OK！那么产生工作的方式是怎么进行的？ 事实上，我们使用 at 这个指令来产生所要运行的工作，并将这个工作以文本文件的方式写入 /var/spool/at/ 目录内，该工作便能等待 atd 这个服务的取用与执行了。就这么简单。

不过，并不是所有的人都可以进行 at 工作调度喔！为什么？因为安全的理由啊～ 很多主机被所谓的“绑架”后，最常发现的就是他们的系统当中多了很多的怪客程序 （cracker program）， 这些程序非常可能运用工作调度来执行或蒐集系统信息，并定时的回报给怪客团体！ 所以啰，除非是你认可的帐号，否则先不要让他们使用 at 吧！那怎么达到使用 at 的列管呢？

我们可以利用 /etc/at.allow 与 /etc/at.deny 这两个文件来进行 at 的使用限制呢！ 加上这两个文件后， at 的工作情况其实是这样的：

1. 先找寻 **/etc/at.allow** 这个文件，写在这个文件中的使用者才能使用 at ，没有在这个文件中的使用者则不能使用 at （即使没有写在 at.deny 当中）；
2. 如果 /etc/at.allow 不存在，就寻找 **/etc/at.deny** 这个文件，若写在这个 at.deny 的使用者则不能使用 at ，而没有在这个 at.deny 文件中的使用者，就可以使用 at 咯；

3. 如果两个文件都不存在，那么只有 root 可以使用 at 这个指令。

通过这个说明，我们知道 /etc/at.allow 是管理较为严格的方式，而 /etc/at.deny 则较为松散（因为帐号没有在该文件中，就能够执行 at 了）。在一般的 distributions 当中，由于假设系统上的所有用户都是可信任的， 因此系统通常会保留一个空的 /etc/at.deny 文件，意思是允许所有人使用 at 指令的意思（您可以自行检查一下该文件）。 不过，万一你不希望有某些使用者使用 at 的话，将那个使用者的帐号写入 /etc/at.deny 即可！ 一个帐号写一行。

15.2.2 实际运行单一工作调度

单一工作调度的进行就使用 at 这个指令啰！这个指令的运行非常简单！将 at 加上一个时间即可！基本的语法如下：

```
[root@study ~]# at [-mldv] TIME
[root@study ~]# at -c 工作号码
选项与参数：
-m  ：当 at 的工作完成后，即使没有输出讯息，亦以 email 通知使用者该工作已完成。
-l  ：at -l 相当于 atq，列出目前系统上面的所有该使用者的 at 调度；
-d  ：at -d 相当于 atrm ，可以取消一个在 at 调度中的工作；
-v  ：可以使用较明显的时间格式列出 at 调度中的工作列表；
-c  ：可以列出后面接的该项工作的实际指令内容。

TIME：时间格式，这里可以定义出“什么时候要进行 at 这项工作”的时间，格式有：
  HH:MM                ex&gt; 04:00
    在今日的 HH:MM 时刻进行，若该时刻已超过，则明天的 HH:MM 进行此工作。
  HH:MM YYYY-MM-DD        ex&gt; 04:00 2015-07-30
    强制规定在某年某月的某一天的特殊时刻进行该工作！
  HH:MM[am&#124;pm] [Month] [Date]    ex&gt; 04pm July 30
    也是一样，强制在某年某月某日的某时刻进行！
  HH:MM[am&#124;pm] + number [minutes&#124;hours&#124;days&#124;weeks]
    ex&gt; now + 5 minutes    ex&gt; 04pm + 3 days
    就是说，在某个时间点“再加几个时间后”才进行。
```

老实说，这个 at 指令的下达最重要的地方在于“时间”的指定了！鸟哥喜欢使用“ now + ... ” 的方式来定义现在过多少时间再进行工作，但有时也需要定义特定的时间点来进行！下面的范例先看看啰！

```
范例一：再过五分钟后，将 /root/.bashrc 寄给 root 自己
[root@study ~]# at now + 5 minutes  &lt;==记得单位要加 s 喔！
at&gt; /bin/mail -s "testing at job" root &lt; /root/.bashrc
at&gt; &lt;EOT&gt;   &lt;==这里输入 [ctrl] + d 就会出现 &lt;EOF&gt; 的字样！代表结束！
job 2 at Thu Jul 30 19:35:00 2015
# 上面这行信息在说明，第 2 个 at 工作将在 2015/07/30 的 19:35 进行！
# 而执行 at 会进入所谓的 at shell 环境，让你下达多重指令等待运行！

范例二：将上述的第 2 项工作内容列出来查阅
[root@study ~]# at -c 2
#!/bin/sh               &lt;==就是通过 bash shell 的啦！
# atrun uid=0 gid=0
# mail root 0
umask 22
....（中间省略许多的环境变量项目）....
cd /etc/cron\.d &#124;&#124; {
         echo 'Execution directory inaccessible' &gt;&2
         exit 1
}
${SHELL:-/bin/sh} &lt;&lt; 'marcinDELIMITER410efc26'
/bin/mail -s "testing at job" root &lt; /root/.bashrc    # 这一行最重要！
marcinDELIMITER410efc26
# 你可以看到指令执行的目录 （/root），还有多个环境变量与实际的指令内容啦！

范例三：由于机房预计于 2015/08/05 停电，我想要在 2015/08/04 23:00 关机？
[root@study ~]# at 23:00 2015-08-04
at&gt; /bin/sync
at&gt; /bin/sync
at&gt; /sbin/shutdown -h now
at&gt; &lt;EOT&gt;
job 3 at Tue Aug  4 23:00:00 2015
# 您瞧瞧！ at 还可以在一个工作内输入多个指令呢！不错吧！
```

事实上，当我们使用 at 时会进入一个 at shell 的环境来让使用者下达工作指令，此时，建议你最好使用绝对路径来下达你的指令，比较不会有问题喔！由于指令的下达与 PATH 变量有关， 同时与当时的工作目录也有关连 （如果有牵涉到文件的话），因此使用绝对路径来下达指令，会是比较一劳永逸的方法。 为什么呢？举例来说，你在 /tmp 下达“ at now ”然后输入“ mail -s "test" root < .bashrc ”， 问一下，那个 .bashrc 的文件会是在哪里？答案是“ /tmp/.bashrc ”！因为 at 在运行时，会跑到当时下达 at 指令的那个工作目录的缘故啊！

有些朋友会希望“我要在某某时刻，在我的终端机显示出 Hello 的字样”，然后就在 at 里面下达这样的信息“ echo "Hello" ”。等到时间到了，却发现没有任何讯息在屏幕上显示，这是啥原因啊？这是因为 at 的执行与终端机环境无关，而所有 standard output/standard error output 都会传送到执行者的 mailbox 去啦！所以在终端机当然看不到任何信息。那怎办？没关系， 可以通过终端机的设备来处理！假如你在 tty1 登陆，则可以使用“ echo "Hello" > /dev/tty1 ”来取代。



Tips 要注意的是，如果在 at shell 内的指令并没有任何的讯息输出，那么 at 默认不会发 email 给执行者的。 如果你想要让 at 无论如何都发一封 email 告知你是否执行了指令，那么可以使用“ at -m 时间格式 ”来下达指令喔！ at 就会传送一个讯息给执行者，而不论该指令执

行有无讯息输出了！

at 有另外一个很棒的优点，那就是“背景执行”的功能了！什么是背景执行啊？很难了解吗？其实与 bash 的 nohup （第十六章） 类似啦！ 鸟哥提我自己的几个例子来给您听听，您就瞭了！

- 离线继续工作的任务：鸟哥初次接触 Unix 为的是要跑空气品质模式， 那是一种大型的程序，这个程序在当时的硬件下面跑，一个案例要跑 3 天！由于鸟哥也要进行其他研究工作，因此常常使用 Windows 98 （你没看错！鸟哥是老人...） 来连线到 Unix 工作站跑那个 3 天的案例！结果你也该知道， Windows 98 连开三天而不死机的概率是很低的～ @_@～ 而死机时，所有在 Windows 上的连线都会中断！包括鸟哥在跑的那个程序也中断了～呜呜～明明再三个钟头就跑完的程序， 由于死机害我又得跑 3 天！
- 另一个常用的时刻则是例如上面的范例三，由于某个突发状况导致你必须要进行某项工作时，这个 at 就很好用啦！

由于 at 工作调度的使用上，系统会将该项 at 工作独立出你的 bash 环境中， 直接交给系统的 atd 程序来接管，因此，当你下达了 at 的工作之后就可以立刻离线了， 剩下的工作就完全交给 Linux 管理即可！所以啰，如果有长时间的网络工作时，嘿嘿！ 使用 at 可以让你免除网络断线后的困扰喔！ ^_^

- at 工作的管理

那么万一我下达了 at 之后，才发现指令输入错误，该如何是好？就将他移除啊！利用 atq 与 atrm 吧！

```
[root@study ~]# atq
[root@study ~]# atrm （jobnumber）

范例一：查询目前主机上面有多少的 at 工作调度？
[root@study ~]# atq
3       Tue Aug  4 23:00:00 2015 a root
# 上面说的是：“在 2015/08/04 的 23:00 有一项工作，该项工作指令下达者为
# root”而且，该项工作的工作号码 （jobnumber） 为 3 号喔！

范例二：将上述的第 3 个工作移除！
[root@study ~]# atrm 3
[root@study ~]# atq
# 没有任何信息，表示该工作被移除了！
```

如此一来，你可以利用 atq 来查询，利用 atrm 来删除错误的指令，利用 at 来直接下达单一工作调度！很简单吧！ 不过，有个问题需要处理一下。如果你是在一个非常忙碌的系统下运行 at ， 能不能指定你的工作在系统较闲的时候才进行呢？可以的，那就使用 batch 指令吧！

- batch：系统有空时才进行背景任务

其实 batch 是利用 at 来进行指令的下达啦！只是加入一些控制参数而已。这个 batch 神奇的地方在于：他会在 CPU 的工作负载小于 0.8 的时候，才进行你所下达的工作任务啦！ 那什么是工作负载 0.8 呢？这个工作负载的意思是： CPU 在单一时间点所负责的工作数量。不是

CPU 的使用率喔！ 举例来说，如果我有一只程序他需要一直使用 CPU 的运算功能，那么此时 CPU 的使用率可能到达 100% ， 但是 CPU 的工作负载则是趋近于“ 1 ”，因为 CPU 仅负责一个工作嘛！如果同时执行这样的程序两支呢？ CPU 的使用率还是 100% ，但是工作负载则变成 2 了！了解乎？

所以也就是说，当 CPU 的工作负载越大，代表 CPU 必须要在不同的工作之间进行频繁的工作切换。 这样的 CPU 运行情况我们在第零章有谈过，忘记的话请回去瞧瞧！因为一直切换工作，所以会导致系统忙碌啊！ 系统如果很忙碌，还要额外进行 at ，不太合理！所以才有 batch 指令的产生！

在 CentOS 7 下面的 batch 已经不再支持时间参数了，因此 batch 可以拿来作为判断是否要立刻执行背景程序的依据！ 我们下面来实验一下 batch 好了！为了产生 CPU 较高的工作负载，因此我们用了 12 章里面计算 pi 的脚本，连续执行 4 次这只程序， 来仿真高负载，然后来玩一玩 batch 的工作现象：

```
范例一：请执行 pi 的计算，然后在系统闲置时，执行 updatdb 的任务
[root@study ~]# echo "scale=100000; 4*a（1）" &#124; bc -lq &
[root@study ~]# echo "scale=100000; 4*a（1）" &#124; bc -lq &
[root@study ~]# echo "scale=100000; 4*a（1）" &#124; bc -lq &
[root@study ~]# echo "scale=100000; 4*a（1）" &#124; bc -lq &
# 然后等待个大约数十秒的时间，之后再来确认一下工作负载的情况！
[root@study ~]# uptime
 19:56:45 up 2 days, 19:54,  2 users,  load average: 3.93, 2.23, 0.96

[root@study ~]# batch
at&gt; /usr/bin/updatedb
at&gt; &lt;EOT&gt;
job 4 at Thu Jul 30 19:57:00 2015

[root@study ~]# date;atq
Thu Jul 30 19:57:47 CST 2015
4       Thu Jul 30 19:57:00 2015 b root
# 可以看得到，明明时间已经超过了，却没有实际执行 at 的任务！

[root@study ~]# jobs
[1]   Running                 echo "scale=100000; 4*a（1）" &#124; bc -lq &
[2]   Running                 echo "scale=100000; 4*a（1）" &#124; bc -lq &
[3]-  Running                 echo "scale=100000; 4*a（1）" &#124; bc -lq &
[4]+  Running                 echo "scale=100000; 4*a（1）" &#124; bc -lq &
[root@study ~]# kill -9 %1 %2 %3 %4
# 这时先用 jobs 找出背景工作，再使用 kill 删除掉四个背景工作后，慢慢等待工作负载的下降

[root@study ~]# uptime; atq
 20:01:33 up 2 days, 19:59,  2 users,  load average: 0.89, 2.29, 1.40
4       Thu Jul 30 19:57:00 2015 b root
[root@study ~]# uptime; atq
 20:02:52 up 2 days, 20:01,  2 users,  load average: 0.23, 1.75, 1.28
# 在 19:59 时，由于 loading 还是高于 0.8，因此 atq 可以看得到 at job 还是持续再等待当中喔！
# 但是到了 20:01 时， loading 降低到 0.8 以下了，所以 atq 就执行完毕啰！
```

使用 uptime 可以观察到 1, 5, 15 分钟的“平均工作负载”量，因为是平均值，所以当我们如上表删除掉四个工作后，工作负载不会立即降低， 需要一小段时间让这个 1 分钟平均值慢慢回复到接近 0 啊！当小于 0.8 之后的“整分钟时间”时，atd 就会将 batch 的工作执行掉了！

什么是“整分钟时间”呢？不论是 at 还是下面要介绍的 crontab，他们最小的时间单位是“分钟”，所以，基本上，他们的工作是“每分钟检查一次”来处理的！ 就是整分 （秒为 0 的时候），这样了解乎？同时，你会发现其实 batch 也是使用 atq/atrm 来管理的！

15.3 循环执行的例行性工作调度

相对于 at 是仅执行一次的工作，循环执行的例行性工作调度则是由 cron （crond） 这个系统服务来控制的。刚刚谈过 Linux 系统上面原本就有非常多的例行性工作，因此这个系统服务是默认启动的。另外， 由于使用者自己也可以进行例行性工作调度，所以啰， Linux 也提供使用者控制例行性工作调度的指令 （crontab）。 下面我们分别来聊一聊啰！

15.3.1 使用者的设置

使用者想要创建循环型工作调度时，使用的是 crontab 这个指令啦～不过，为了安全性的问题， 与 at 同样的，我们可以限制使用 crontab 的使用者帐号喔！使用的限制数据有：

- /etc/cron.allow： 将可以使用 crontab 的帐号写入其中，若不在这个文件内的使用者则不可使用 crontab；
- /etc/cron.deny： 将不可以使用 crontab 的帐号写入其中，若未记录到这个文件当中的使用者，就可以使用 crontab 。

与 at 很像吧！同样的，以优先顺序来说， /etc/cron.allow 比 /etc/cron.deny 要优先， 而判断上面，这两个文件只选择一个来限制而已，因此，建议你只要保留一个即可， 免得影响自己在设置上面的判断！一般来说，系统默认是保留 /etc/cron.deny ， 你可以将不想让他执行 crontab 的那个使用者写入 /etc/cron.deny 当中，一个帐号一行！

当使用者使用 crontab 这个指令来创建工作调度之后，该项工作就会被纪录到 /var/spool/cron/ 里面去了，而且是以帐号来作为判别的喔！举例来说， dmtsai 使用 crontab 后， 他的工作会被纪录到 /var/spool/cron/dmtsai 里头去！但请注意，不要使用 vi 直接编辑该文件， 因为可能由于输入语法错误，会导致无法执行 cron 喔！另外， cron 执行的每一项工作都会被纪录到 /var/log/cron 这个登录文件中，所以啰，如果你的 Linux 不知道有否被植入木马时，也可以搜寻一下 /var/log/cron 这个登录文件呢！

好了，那么我们就来聊一聊 crontab 的语法吧！

```
[root@study ~]# crontab [-u username] [-l&#124;-e&#124;-r]
选项与参数：
-u  ：只有 root 才能进行这个任务，亦即帮其他使用者创建/移除 crontab 工作调度；
-e  ：编辑 crontab 的工作内容
-l  ：查阅 crontab 的工作内容
-r  ：移除所有的 crontab 的工作内容，若仅要移除一项，请用 -e 去编辑。

范例一：用 dmtsai 的身份在每天的 12:00 发信给自己
[dmtsai@study ~]$ crontab -e
# 此时会进入 vi 的编辑画面让您编辑工作！注意到，每项工作都是一行。
0   12  *  *  * mail -s "at 12:00" dmtsai &lt; /home/dmtsai/.bashrc
#分 时 日 月 周 &#124;&lt;==============指令串========================&gt;&#124;
```

默认情况下，任何使用者只要不被列入 /etc/cron.deny 当中，那么他就可以直接下达“ crontab -e ”去编辑自己的例行性命令了！整个过程就如同上面提到的，会进入 vi 的编辑画面， 然后以一个工作一行来编辑，编辑完毕之后输入“ :wq ”储存后离开 vi 就可以了！ 而每项工作 （每行） 的格式都是具有六个字段，这六个字段的意义为：

代表意义	分钟	小时	日期	月份	周	指令
数字范围	0-59	0-23	1-31	1-12	0-7	呀就指令啊

比较有趣的是那个“周”喔！周的数字为 0 或 7 时，都代表“星期天”的意思！另外，还有一些辅助的字符，大概有下面这些：

特殊字符	代表意义
*（星号）	代表任何时刻都接受的意思！举例来说，范例一内那个日、月、周都是 * ， 就代表着“不论何月、何日的礼拜几的 12:00 都执行后续指令”的意思！
,（逗号）	代表分隔时段的意思。举例来说，如果要下达的工作是 3:00 与 6:00 时，就会是： `> 0 3,6 * * * command` 时间参数还是有五栏，不过第二栏是 3,6 ，代表 3 与 6 都适用！
-（减号）	代表一段时间范围内，举例来说， 8 点到 12 点之间的每小时的 20 分都进行一项工作： `> 20 8-12 * * * command` 仔细看到第二栏变成 8-12 喔！代表 8,9,10,11,12 都适用的意思！
/n（斜线）	那个 n 代表数字，亦即是“每隔 n 单位间隔”的意思，例如每五分钟进行一次，则： `> */5 * * * * command` 很简单吧！用 * 与 /5 来搭配，也可以写成 0-59/5 ，相同意思！

我们就来搭配几个例子练习看看吧！下面的案例请实际用 dmtsai 这个身份作看看喔！后续的动作才能够搭配起来！

例题：假若你的女朋友生日是 5 月 2 日，你想要在 5 月 1 日的 23:59 发一封信给他，这封信的内容已经写在 /home/dmtsai/lover.txt 内了，该如何进行？答：直接下达 crontab -e 之后，编辑成为：

```
> 59 23 1 5 * mail kiki &lt; /home/dmtsai/lover.txt
```

那样的话，每年 kiki 都会收到你的这封信喔！（当然啰，信的内容就要每年变一变啦！）

例题：假如每五分钟需要执行 /home/dmtsai/test.sh 一次，又该如何？答：同样使用 crontab -e 进入编辑：

```
> */5 * * * * /home/dmtsai/test.sh
```

那个 crontab 每个人都只有一个文件存在，就是在 /var/spool/cron 里面啊！ 还有建议您：“指令下达时，最好使用绝对路径，这样比较不会找不到可执行文件喔！”

例题：假如你每星期六都与朋友有约，那么想要每个星期五下午 4:30 告诉你朋友星期六的约会不要忘记，则：答：还是使用 crontab -e 啊！

```
> 30 16 * * 5 mail friend@his.server.name &lt; /home/dmtsai/friend.txt
```

眞的是很简单吧！呵呵！那么，该如何查询使用者目前的 crontab 内容呢？我们可以这样来看看：

```
[dmtsai@study ~]$ crontab -l
0 12 * * * mail -s "at 12:00" dmtsai &lt; /home/dmtsai/.bashrc
59 23 1 5 * mail kiki &lt; /home/dmtsai/lover.txt
*/5 * * * * /home/dmtsai/test.sh
30 16 * * 5 mail friend@his.server.name &lt; /home/dmtsai/friend.txt

# 注意，若仅想要移除一项工作而已的话，必须要用 crontab -e 去编辑～
# 如果想要全部的工作都移除，才使用 crontab -r 喔！
[dmtsai@study ~]$ crontab -r
[dmtsai@study ~]$ crontab -l
no crontab for dmtsai
```

看到了吗？ crontab “整个内容都不见了！”所以请注意：“如果只是要删除某个 crontab 的工作项目，那么请使用 crontab -e 来重新编辑即可！”如果使用 -r 的参数，是会将所有的 crontab 数据内容都删掉的！千万注意了！

15.3.2 系统的配置文件： /etc/crontab, /etc/cron.d/*

这个“ crontab -e ”是针对使用者的 cron 来设计的，如果是“系统的例行性任务”时， 该怎么办呢？是否还是需要以 crontab -e 来管理你的例行性工作调度呢？当然不需要，你只要编辑 /etc/crontab 这个文件就可以啦！有一点需要特别注意喔！那就是 crontab -e 这个 crontab 其实是 /usr/bin/crontab 这个可执行文件，但是 /etc/crontab 可是一个“纯文本文件”喔！你可以 root 的身份编辑一下这个文件哩！

基本上， cron 这个服务的最低侦测限制是“分钟”，所以“ cron 会每分钟去读取一次 /etc/crontab 与 /var/spool/cron 里面的数据内容 ”，因此，只要你编辑完 /etc/crontab 这个文件，并且将他储存之后，那么 cron 的设置就自动的会来执行了！



Tips 在 Linux 下面的 crontab 会自动的帮我们每分钟重新读取一次 /etc/crontab 的例行工作事项，但是某些原因或者是其他的 Unix 系统中，由于 crontab 是读到内存当中的，所以在你修改完 /etc/crontab 之后，可能并不会马上执行， 这个时候请重新启动 crond 这个服务吧！“systemctl restart crond”

废话少说，我们就来看一下这个 /etc/crontab 的内容吧！

```
[root@study ~]# cat /etc/crontab
SHELL=/bin/bash                        &lt;==使用哪种 shell 接口
PATH=/sbin:/bin:/usr/sbin:/usr/bin     &lt;==可执行文件搜寻路径
MAILTO=root                            &lt;==若有额外STDOUT，以 email将数据送给谁

# Example of job definition:
# .---------------- minute （0 - 59）
# &#124;  .------------- hour （0 - 23）
# &#124;  &#124;  .---------- day of month （1 - 31）
# &#124;  &#124;  &#124;  .------- month （1 - 12） OR jan,feb,mar,apr ...
# &#124;  &#124;  &#124;  &#124;  .---- day of week （0 - 6） （Sunday=0 or 7） OR sun,mon
# &#124;  &#124;  &#124;  &#124;  &#124;
# *  *  *  *  * user-name  command to be executed
```

看到这个文件的内容你大概就了解了吧！呵呵，没错！这个文件与将刚刚我们下达 crontab -e 的内容几乎完全一模一样！只是有几个地方不太相同：

- MAILTO=root：

这个项目是说，当 /etc/crontab 这个文件中的例行性工作的指令发生错误时，或者是该工作的执行结果有 STDOUT/STDERR 时，会将错误讯息或者是屏幕显示的讯息传给谁？默认当然是由系统直接寄发一封 mail 给 root 啦！不过， 由于 root 并无法在用户端中以 POP3 之类的软件收信，因此，鸟哥通常都将这个 e-mail 改成自己的帐号，好让我随时了解系统的状况！例如： MAILTO=dmtsai@my.host.name

- PATH=....：

还记得我们在第十章的 BASH 当中一直提到的可执行文件路径问题吧！ 没错啦！这里就是输入可执行文件的搜寻路径！使用默认的路径设置就已经很足够了！

- “分 时 日 月 周 身份 指令”七个字段的设置

这个 /etc/crontab 里面可以设置的基本语法与 crontab -e 不太相同喔！前面同样是分、时、日、月、周五个字段， 但是在五个字段后面接的并不是指令，而是一个新的字段，那就是“执行后面那串指令的身份”为何！这与使用者的 crontab -e 不相同。由于使用者自己的 crontab 并不需要指定身份，但 /etc/crontab 里面当然要指定身份啦！以上表的内容来说，系统默认的例行性工作是以 root 的身份来进行的。

- crond 服务读取配置文件的位置

一般来说，crond 默认有三个地方会有执行脚本配置文件，他们分别是：

- /etc/crontab
- /etc/cron.d/*
- /var/spool/cron/*

这三个地方中，跟系统的运行比较有关系的两个配置文件是放在 /etc/crontab 文件内以及 /etc/cron.d/* 目录内的文件， 另外一个是跟用户自己的工作比较有关的配置文件，就是放在 /var/spool/cron/ 里面的文件群。 现在我们已经知道了 /var/spool/cron 以及 /etc/crontab 的内

容，那现在来瞧瞧 /etc/cron.d 里面的东西吧！

```
[root@study ~]# ls -l /etc/cron.d
-rw-r--r--. 1 root root 128 Jul 30  2014 0hourly
-rw-r--r--. 1 root root 108 Mar  6 10:12 raid-check
-rw-------. 1 root root 235 Mar  6 13:45 sysstat
-rw-r--r--. 1 root root 187 Jan 28  2014 unbound-anchor
# 其实说真的，除了 /etc/crontab 之外，crond 的配置文件还不少耶！上面就有四个设置！
# 先让我们来瞧瞧 0hourly 这个配置文件的内容吧！

[root@study ~]# cat /etc/cron.d/0hourly
# Run the hourly jobs
SHELL=/bin/bash
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
01 * * * * root run-parts /etc/cron.hourly
# 瞧一瞧，内容跟 /etc/crontab 几乎一模一样！但实际上是有设置值喔！就是最后一行！
```

如果你想要自己开发新的软件，该软件要拥有自己的 crontab 定时指令时，就可以将“分、时、日、月、周、身份、指令”的配置文件放置到 /etc/cron.d/ 目录下！ 在此目录下的文件是“crontab 的配置文件脚本”。



Tips 以鸟哥来说，现在鸟哥有在开发一些虚拟化教室的软件，该软件需要定时清除一些垃圾防火墙规则， 那鸟哥就是将要执行的时间与指令设计好，然后直接将设置写入到 /etc/cron.d/newfile 即可！未来如果这个软件要升级， 直接将该文件覆盖成新文件即可！比起手动去分析 /etc/crontab 要单纯的多！

另外，请注意一下上面表格中提到的最后一行，每个整点的一分会执行“ run-parts /etc/cron.hourly ”这个指令～咦！那什么是 run-parts 呢？ 如果你有去分析一下这个可执行文件，会发现他就是 shell script，run-parts 脚本会在大约 5 分钟内随机选一个时间来执行 /etc/cron.hourly 目录内的所有可执行文件！因此，放在 /etc/cron.hourly/ 的文件，必须是能被直接执行的指令脚本， 而不是分、时、日、月、周的设置值喔！注意注意！

也就是说，除了自己指定分、时、日、月、周加上指令路径的 crond 配置文件之外，你也可以直接将指令放置到（或链接到）/etc/cron.hourly/ 目录下， 则该指令就会被 crond 在每小时的 1 分开始后的 5 分钟内，随机取一个时间点来执行啰！你无须手动去指定分、时、日、月、周就是了。

但是眼尖的朋友可能还会发现，除了可以直接将指令放到 /etc/cron.hourly/ 让系统每小时定时执行之外，在 /etc/ 下面其实还有 /etc/cron.daily/, /etc/cron.weekly/, /etc/cron.monthly/，那三个目录是代表每日、每周、每月各执行一次的意思吗？嘿嘿！ 厉害喔！没错～是这样～不过，跟 /etc/cron.hourly/ 不太一样的是，那三个目录是由 anacron 所执行的，而 anacron 的执行方式则是放在 /etc/cron.hourly/0anacron 里面耶～跟前几代 anacron 是单独的 service 不太一样喔！这部份留待下个小节再来讨论。

最后，让我们总结一下吧：

- 个人化的行为使用“ crontab -e ”：如果你是依据个人需求来创建的例行工作调度，建议直接使用 crontab -e 来创建你的工作调度较佳！ 这样也能保障你的指令行为不会被大家看到 （/etc/crontab 是大家都能读取的权限喔！）；
- 系统维护管理使用“ vim /etc/crontab ”：如果你这个例行工作调度是系统的重要工作，为了让自己管理方便，同时容易追踪，建议直接写入 /etc/crontab 较佳！
- 自己开发软件使用“ vim /etc/cron.d/newfile ”：如果你是想要自己开发软件，那当然最好就是使用全新的配置文件，并且放置于 /etc/cron.d/ 目录内即可。
- 固定每小时、每日、每周、每天执行的特别工作：如果与系统维护有关，还是建议放置到 /etc/crontab 中来集中管理较好。 如果想要偷懒，或者是一定要再某个周期内进行的任务，也可以放置到上面谈到的几个目录中，直接写入指令即可！

15.3.3 一些注意事项

有的时候，我们以系统的 cron 来进行例行性工作的创建时，要注意一些使用方面的特性。 举例来说，如果我们有四个工作都是五分钟要进行一次的，那么是否这四个动作全部都在同一个时间点进行？ 如果同时进行，该四个动作又很耗系统资源，如此一来，每五分钟的某个时刻不是会让系统忙得要死？ 呵呵！此时好好的分配一些执行时间就 OK 啦！所以，注意一下：

- 资源分配不均的问题

当大量使用 crontab 的时候，总是会有问题发生的，最严重的问题就是“系统资源分配不均”的问题， 以鸟哥的系统为例，我有侦测主机流量的信息，包括：

- 流量
- 区域内其他 PC 的流量侦测
- CPU 使用率
- RAM 使用率
- 线上人数实时侦测

如果每个流程都在同一个时间启动的话，那么在某个时段时，我的系统会变的相当的繁忙，所以，这个时候就必须要分别设置啦！我可以这样做：

```
[root@study ~]# vim /etc/crontab
1,6,11,16,21,26,31,36,41,46,51,56 * * * * root  CMD1
2,7,12,17,22,27,32,37,42,47,52,57 * * * * root  CMD2
3,8,13,18,23,28,33,38,43,48,53,58 * * * * root  CMD3
4,9,14,19,24,29,34,39,44,49,54,59 * * * * root  CMD4
```

看到了没？那个“ , ”分隔的时候，请注意，不要有空白字符！（连续的意思）如此一来， 则可以将每五分钟工作的流程分别在不同的时刻来工作！则可以让系统的执行较为顺畅呦！

- 取消不要的输出项目

另外一个困扰发生在“ 当有执行成果或者是执行的项目中有输出的数据时，该数据将会 mail 给 MAILTO 设置的帐号 ”，好啦，那么当有一个调度一直出错（例如 DNS 的侦测系统当中，若 DNS 上层主机挂掉，那么你就会一直收到错误讯息！）怎么办？呵呵！还记得第十章谈到的数据流重导向吧？ 直接以“数据流重导向”将输出的结果输出到 /dev/null 这个垃圾桶当中就好了！

- 安全的检验

很多时候被植入木马都是以例行命令的方式植入的，所以可以借由检查 /var/log/cron 的内容来视察是否有“非您设置的 cron 被执行了？”这个时候就需要小心一点啰！

- 周与日月不可同时并存

另一个需要注意的地方在于：“你可以分别以周或者是日月为单位作为循环，但你不可使用「几月几号且为星期几」的模式工作”。 这个意思是说，你不可以这样编写一个工作调度：

```
30 12 11 9 5 root echo "just test"   &lt;==这是错误的写法
```

本来你以为九月十一号且为星期五才会进行这项工作，无奈的是，系统可能会判定每个星期五作一次，或每年的 9 月 11 号分别进行，如此一来与你当初的规划就不一样了～所以啰，得要注意这个地方！



Tips 根据某些人的说法，这个月日、周不可并存的问题已经在新版中被克服了～不过，鸟哥并没有实际去验证他！目前也不打算验证他！ 因为，周就是周，月日就月日，单一执行点就单一执行点，无须使用 crontab 去设置固定的日期啊！您说是吧？

15.4 可唤醒停机期间的工作任务

想像一个环境，你的 Linux 服务器有一个工作是需要在每周的星期天凌晨 2 点进行，但是很不巧的，星期六停电了～所以你得要星期一才能进公司去启动服务器。 那么请问，这个星期天的工作调度还要不要进行？因为你开机的时候已经是星期一，所以星期天的工作当然不会被进行，对吧！

问题是，若是该工作非常重要 （例如例行备份）， 所以其实你还是希望在下个星期天之前的某天还是进行一下比较好～那你该怎办？自己手动执行？如果你跟鸟哥一样是个记忆力超差的家伙，那么肯定“记不起来某个重要工作要进行”的啦！ 这时候就得要靠 anacron 这个指令的功能了！这家伙可以主动帮你进行时间到了但却没有执行的调度喔！

15.4.1 什么是 anacron

anacron 并不是用来取代 crontab 的，anacron 存在的目的就在于我们上头提到的，在处理非 24 小时一直启动的 Linux 系统的 crontab 的执行！ 以及因为某些原因导致的超过时间而没有被执行的调度工作。

其实 anacron 也是每个小时被 crond 执行一次，然后 anacron 再去检测相关的调度任务有没有被执行，如果有超过期限的工作在， 就执行该调度任务，执行完毕或无须执行任何调度时，anacron 就停止了。

由于 anacron 默认会以一天、七天、一个月为期去侦测系统未进行的 crontab 任务，因此对于某些特殊的使用环境非常有帮助。 举例来说，如果你的 Linux 主机是放在公司给同仁使用的，因为周末假日大家都不在所以也没有必要打开， 因此你的 Linux 是周末都会关机两天的。但是 crontab 大多在每天的凌晨以及周日的早上进行各项任务， 偏偏你又关机了，此时系统很多 crontab 的任务就无法进行。 anacron 刚好可以解决这个问题！

那么 anacron 又是怎么知道我们的系统啥时关机的呢？这就得要使用 anacron 读取的时间记录文件 （timestamps） 了！ anacron 会去分析现在的时间与时间记录文件所记载的上次执行 anacron 的时间，两者比较后若发现有差异， 那就是在某些时刻没有进行 crontab 啰！此时 anacron 就会开始执行未进行的 crontab 任务了！

15.4.2 anacron 与 /etc/anacrontab

anacron 其实是一支程序并非一个服务！这支程序在 CentOS 当中已经进入 crontab 的调度喔！同时 anacron 会每个小时被主动执行一次喔！ 咦！每个小时？所以 anacron 的配置文件应该放置在 /etc/cron.hourly 吗？嘿嘿！您真内行～赶紧来瞧一瞧：

```
[root@study ~]# cat /etc/cron.hourly/0anacron
#!/bin/sh
# Check whether 0anacron was run today already
if test -r /var/spool/anacron/cron.daily; then
    day=`cat /var/spool/anacron/cron.daily`
fi
if [ `date +%Y%m%d` = "$day" ]; then
    exit 0;
fi
# 上面的语法在检验前一次执行 anacron 时的时间戳记！

# Do not run jobs when on battery power
if test -x /usr/bin/on_ac_power; then
    /usr/bin/on_ac_power &gt;/dev/null 2&gt;&1
    if test $? -eq 1; then
    exit 0
    fi
fi
/usr/sbin/anacron -s
# 所以其实也仅是执行 anacron -s 的指令！因此我们得来谈谈这支程序！
```

基本上， anacron 的语法如下：

```
[root@study ~]# anacron [-sfn] [job]..
[root@study ~]# anacron -u [job]..
选项与参数：
-s  ：开始一连续的执行各项工作 （job），会依据时间记录文件的数据判断是否进行；
-f  ：强制进行，而不去判断时间记录文件的时间戳记；
-n  ：立刻进行未进行的任务，而不延迟 （delay） 等待时间；
-u  ：仅更新时间记录文件的时间戳记，不进行任何工作。
job ：由 /etc/anacrontab 定义的各项工作名称。
```

在我们的 CentOS 中，anacron 的进行其实是在每个小时都会被抓出来执行一次， 但是为了担心 anacron 误判时间参数，因此 /etc/cron.hourly/ 里面的 anacron 才会在文件名之前加个 0（0anacron），让 anacron 最先进行！就是为了让时间戳记先更新！以避免 anacron 误判 crontab 尚未进行任何工作的意思。

接下来我们看一下 anacron 的配置文件： /etc/anacrontab 的内容好了：

```
[root@study ~]# cat /etc/anacrontab
SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root
RANDOM_DELAY=45           # 随机给予最大延迟时间，单位是分钟
START_HOURS_RANGE=3-22    # 延迟多少个小时内应该要执行的任务时间

1         5       cron.daily         nice run-parts /etc/cron.daily
7        25       cron.weekly        nice run-parts /etc/cron.weekly
@monthly 45       cron.monthly       nice run-parts /etc/cron.monthly
天数      延迟时间   工作名称定义         实际要进行的指令串
# 天数单位为天；延迟时间单位为分钟；工作名称定义可自订，指令串则通常与 crontab 的设置相同！

[root@study ~]# more /var/spool/anacron/*
::::::::::::::
/var/spool/anacron/cron.daily
::::::::::::::
20150731
::::::::::::::
/var/spool/anacron/cron.monthly
::::::::::::::
20150703
::::::::::::::
/var/spool/anacron/cron.weekly
::::::::::::::
20150727
# 上面则是三个工作名称的时间记录文件以及记录的时间戳记
```

我们拿 /etc/cron.daily/ 那一行的设置来说明好了。那四个字段的意义分别是：

- 天数：anacron 执行当下与时间戳记 （/var/spool/anacron/ 内的时间纪录档） 相差的天数，若超过此天数，就准备开始执行，若没有超过此天数，则不予执行后续的指令。
- 延迟时间：若确定超过天数导致要执行调度工作了，那么请延迟执行的时间，因为担心立即启动会有其他资源冲突的问题吧！
- 工作名称定义：这个没啥意义，就只是会在 /var/log/cron 里头记载该项任务的名称这样！通常与后续的目录资源名称相同即可。
- 实际要进行的指令串：有没有跟 0hourly 很像啊！没错！相同的作法啊！通过 run-parts 来处理的！

根据上面的配置文件内容，我们大概知道 anacron 的执行流程应该是这样的 （以 cron.daily 为例） ：

1. 由 /etc/anacrontab 分析到 cron.daily 这项工作名称的天数为 1 天；
2. 由 /var/spool/anacron/cron.daily 取出最近一次执行 anacron 的时间戳记；
3. 由上个步骤与目前的时间比较，若差异天数为 1 天以上 （含 1 天） ，就准备进行指令；
4. 若准备进行指令，根据 /etc/anacrontab 的设置，将延迟 5 分钟 + 3 小时 （看 START_HOURS_RANGE 的设置） ；
5. 延迟时间过后，开始执行后续指令，亦即“ run-parts /etc/cron.daily ”这串指令；
6. 执行完毕后， anacron 程序结束。

如此一来，放置在 /etc/cron.daily/ 内的任务就会在一天后一定会被执行的！因为 anacron 是每个小时被执行一次嘛！ 所以，现在你知道为什么隔了一阵子才将 CentOS 开机，开机过后约 1 小时左右系统会有一小段时间的忙碌！而且硬盘会跑个不停！那就是因为 anacron 正在

执行过去 /etc/cron.daily/, /etc/cron.weekly/, /etc/cron.monthly/ 里头的未进行的各项工作调度啦！这样对 anacron 有没有概念了呢？ ^_^

最后，我们来总结一下本章谈到的许多配置文件与目录的关系吧！这样我们才能了解 crond 与 anacron 的关系：

1. crond 会主动去读取 /etc/crontab, /var/spool/cron/, */etc/cron.d/* 等配置文件，并依据“分、时、日、月、周”的时间设置去各项工作调度；
2. 根据 /etc/cron.d/0hourly 的设置，主动去 /etc/cron.hourly/ 目录下，执行所有在该目录下的可执行文件；
3. 因为 /etc/cron.hourly/0anacron 这个指令档的缘故，主动的每小时执行 anacron ，并调用 /etc/anacrontab 的配置文件；
4. 根据 /etc/anacrontab 的设置，依据每天、每周、每月去分析 /etc/cron.daily/, /etc/cron.weekly/, /etc/cron.monthly/ 内的可执行文件，以进行固定周期需要执行的指令。

也就是说，如果你每个周日的需要执行的动作是放置于 /etc/crontab 的话，那么该动作只要过期了就过期了，并不会被抓回来重新执行。但如果是放置在 /etc/cron.weekly/ 目录下，那么该工作就会定期，几乎一定会在一周内执行一次～如果你关机超过一周，那么一开机后的数个小时内，该工作就会主动的被执行喔！ 真的吗？对啦！因为 /etc/anacrontab 的定义啦！



Tips 基本上，crontab 与 at 都是“定时”去执行，过了时间就过了！不会重新来一遍～那 anacron 则是“定期”去执行，某一段周期的执行～ 因此，两者可以并行，并不会互相冲突啦！

15.5 重点回顾

- 系统可以通过 at 这个指令来调度单一工作的任务！“at TIME”为指令下达的方法，当 at 进入调度后， 系统执行该调度工作时，会到下达时的目录进行任务；
- at 的执行必须要有 atd 服务的支持，且 /etc/at.deny 为控制是否能够执行的使用者帐号；
- 通过 atq, atrm 可以查询与删除 at 的工作调度；
- batch 与 at 相同，不过 batch 可在 CPU 工作负载小于 0.8 时才进行后续的工作调度；
- 系统的循环例行性工作调度使用 crond 这个服务，同时利用 crontab -e 及 /etc/crontab 进行调度的安排；
- crontab -e 设置项目分为六栏，“分、时、日、月、周、指令”为其设置依据；
- /etc/crontab 设置分为七栏，“分、时、日、月、周、执行者、指令”为其设置依据；
- anacron 配合 /etc/anacrontab 的设置，可以唤醒停机期间系统未进行的 crontab 任务！

15.6 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ） 简答题：

- 今天假设我有一个指令程序，名称为： ping.sh 这个文件名！我想要让系统每三分钟执行这个文件一次， 但是偏偏这个文件会有很多的讯息显示出来，所以我的 root 帐号每天都会收到差不多四百多封的信件，光是收信就差不多快要疯掉了！ 那么请问应该怎么设置比较好呢？这个涉及数据流重导向的问题，我们可以将他导入文件或者直接丢弃！如果该讯息不重要的话， 那么就予以丢弃，如果讯息很重要的话，才将他保留下来！假设今天这个命令不重要， 所以将他丢弃掉！因此，可以这样写：

  > /3 * root /usr/local/ping.sh > /dev/null 2>&1

- 您预计要在 2016 年的 2 月 14 日寄出一封给 kiki ，只有该年才寄出！该如何下达指令？ at 1am 2016-02-14

- 下达 crontab -e 之后，如果输入这一行，代表什么意思？
  - 15 1-5 /usr/local/bin/tea_time.sh在每星期的 1~5 ，下午 3 点的每分钟，共进行 60 次 /usr/local/bin/tea_time.sh 这个文件。 要特别注意的是，每个星期 1~5 的 3 点都会进行 60 次ㄟ！很麻烦吧～是错误的写法啦～ 应该是要写成： 30 15 1-5 /usr/local/bin/tea_time.sh
- 我用 vi 编辑 /etc/crontab 这个文件，我编辑的那一行是这样的： 25 00 0 /usr/local/bin/backup.sh 这一行代表的意义是什么？这一行代表......没有任何意义！因为语法错误！您必须要了解，在 /etc/crontab 当中每一行都必须要有使用者才行！所以，应该要将原本那行改成： 25 00 0 root /usr/local/bin/backup.sh
- 请问，您的系统每天、每周、每个月各有进行什么工作？因为 CentOS 系统默认的例行性命令都放置在 /etc/cron.* 里面，所以，你可以自行去： /etc/cron.daily/, /etc/cron.week/, /etc/cron.monthly/ 这三个目录内看一看， 就知道啦！ ^_^
- 每个星期六凌晨三点去系统搜寻一下内有 SUID/SGID 的任何文件！并将结果输出到 /tmp/uidgid.filesvi /etc/crontab 0 3 6 root find / -perm /6000 > /tmp/uidgid.files

2002/05/30：第一次完成 2003/02/10：重新编排与加入 FAQ 2005/09/07：将旧的文章移动到此处 。 2005/09/07：呼呼！终于完成风格啰～同时加入一些习题练习。 2009/03/12：将旧的文件移动到此处。 2009/03/14：加入 batch 这个项目的说明！与 at 有关！ 2009/03/15：加入了 anacron 这玩意的简单说明！ 2009/09/11：稍微修订一下说明语气与链接数据。 2015/07/31：将旧的基于 CentOS5 的版本移动到这里。

第十六章、程序管理与 SELinux 初探

最近更新日期：20//

一个程序被载入到内存当中运行，那么在内存内的那个数据就被称为程序（process）。程序是操作系统上非常重要的概念， 所有系统上面跑的数据都会以程序的型态存在。那么系统的程序有哪些状态？不同的状态会如何影响系统的运行？ 程序之间是否可以互相控管等等的，这些都是我们所必须要知道的项目。 另外与程序有关的还有 SELinux 这个加强文件存取安全性的咚咚，也必须要做个了解呢！

16.1 什么是程序 （process）

由前面一连几个章节的数据看来，我们一直强调在 Linux 下面所有的指令与你能够进行的动作都与权限有关， 而系统如何判定你的权限呢？当然就是第十三章帐号管理当中提到的 UID/GID 的相关概念，以及文件的属性相关性啰！再进一步来解释，你现在大概知道，在 Linux 系统当中：“触发任何一个事件时，系统都会将他定义成为一个程序，并且给予这个程序一个 ID ，称为 PID，同时依据启发这个程序的使用者与相关属性关系，给予这个 PID 一组有效的权限设置。” 从此以后，这个 PID 能够在系统上面进行的动作，就与这个 PID 的权限有关了！

看这个定义似乎没有什么很奇怪的地方，不过，您得要了解什么叫做“触发事件”才行啊！ 我们在什么情况下会触发一个事件？而同一个事件可否被触发多次？呵呵！来了解了解先！

16.1.1 程序与程序 （process & program）

我们如何产生一个程序呢？其实很简单啦，就是“执行一个程序或指令”就可以触发一个事件而取得一个 PID 啰！我们说过，系统应该是仅认识 binary file 的，那么当我们要让系统工作的时候，当然就是需要启动一个 binary file 啰，那个 binary file 就是程序 （program） 啦！

那我们知道，每个程序都有三组人马的权限，每组人马都具有 r/w/x 的权限，所以：“不同的使用者身份执行这个 program 时，系统给予的权限也都不相同！”举例来说，我们可以利用 touch 来创建一个空的文件，当 root 执行这个 touch 指令时，他取得的是 UID/GID = 0/0 的权限，而当 dmtsai （UID/GID=501/501） 执行这个 touch 时，他的权限就跟 root 不同啦！我们将这个概念绘制成图示来瞧瞧如下：

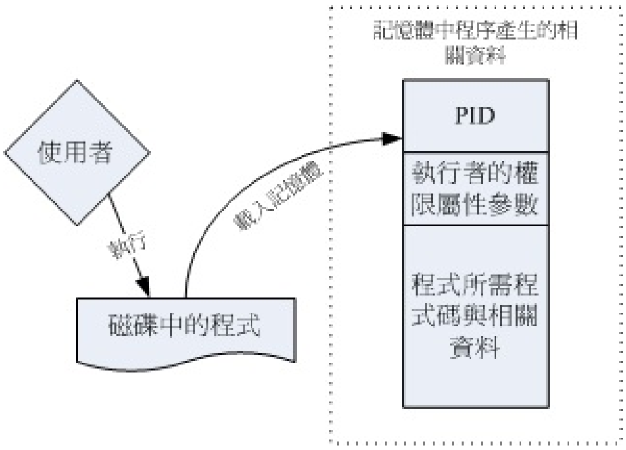


图16.1.1、程序被载入成为程序以及相关数据的示意图

如上图所示，程序一般是放置在实体磁盘中，然后通过使用者的执行来触发。触发后会载入到内存中成为一个个体，那就是程序。 为了操作系统可管理这个程序，因此程序有给予执行者的权限/属性等参数，并包括程序所需要的指令码与数据或文件数据等， 最后再给予一个

PID 。系统就是通过这个 PID 来判断该 process 是否具有权限进行工作的！他是很重要的哩！

举个更常见的例子，我们要操作系统的时候，通常是利用连线程序或者直接在主机前面登陆，然后取得我们的 shell 对吧！那么，我们的 shell 是 bash 对吧，这个 bash 在 /bin/bash 对吧，那么同时间的每个人登陆都是执行 /bin/bash 对吧！不过，每个人取得的权限就是不同！也就是说，我们可以这样看：

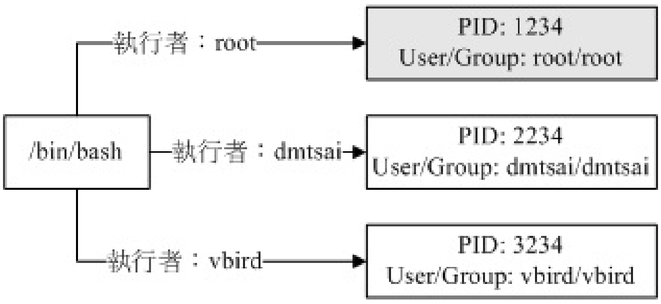


图16.1.2、程序与程序之间的差异

也就是说，当我们登陆并执行 bash 时，系统已经给我们一个 PID 了，这个 PID 就是依据登陆者的 UID/GID （/etc/passwd） 来的啦～以上面的图 16.1.2 配合图 16.1.1 来做说明的话，我们知道 /bin/bash 是一个程序 （program），当 dmtsai 登陆后，他取得一个 PID 号码为 2234 的程序，这个程序的 User/Group 都是 dmtsai ，而当这个程序进行其他作业时，例如上面提到的 touch 这个指令时， 那么由这个程序衍生出来的其他程序在一般状态下，也会沿用这个程序的相关权限的！

让我们将程序与程序作个总结：

- 程序 （program）：通常为 binary program ，放置在储存媒体中 （如硬盘、光盘、软盘、磁带等）， 为实体文件的型态存在；
- 程序 （process）：程序被触发后，执行者的权限与属性、程序的程序码与所需数据等都会被载入内存中， 操作系统并给予这个内存内的单元一个识别码 （PID），可以说，程序就是一个正在运行中的程序。
- 子程序与父程序：

在上面的说明里面，我们有提到所谓的“衍生出来的程序”，那是个啥咚咚？这样说好了，当我们登陆系统后，会取得一个 bash 的 shell ，然后，我们用这个 bash 提供的接口去执行另一个指令，例如 /usr/bin/passwd 或者是 touch 等等，那些另外执行的指令也会被触发成为 PID ，呵呵！那个后来执行指令才产生的 PID 就是“子程序”了，而在我们原本的 bash 环境下，就称为“父程序”了！借用我们在第十章 Bash 谈到的 export 所用的图示好了：

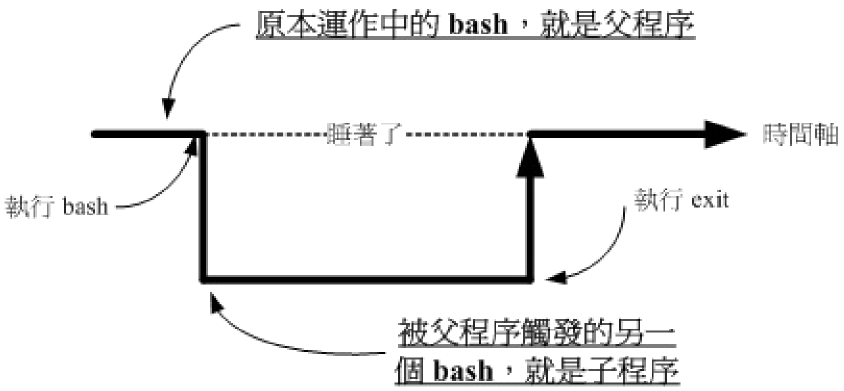


图16.1.3、程序相关系之示意图

所以你必须要知道，程序彼此之间是有相关性的！以上面的图示来看，连续执行两个 bash 后，第二个 bash 的父程序就是前一个 bash。因为每个程序都有一个 PID ，那某个程序的父程序该如何判断？就通过 Parent PID （PPID） 来判断即可。此外，由第十章的 export 内容我们也探讨过环境变量的继承问题，子程序可以取得父程序的环境变量啦！ 让我们来进行下面的练习，以了解什么是子程序/父程序。

例题：请在目前的 bash 环境下，再触发一次 bash ，并以“ ps -l ”这个指令观察程序相关的输出信息。答：直接执行 bash ，会进入到子程序的环境中，然后输入 ps -l 后，出现：

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000 13928 13927  0  80   0 - 29038 wait   pts/0    00:00:00 bash
0 S  1000 13970 13928  1  80   0 - 29033 wait   pts/0    00:00:00 bash
0 R  1000 14000 13970  0  80   0 - 30319 -      pts/0    00:00:00 ps
```

有看到那个 PID 与 PPID 吗？第一个 bash 的 PID 与第二个 bash 的 PPID 都是 13928 啊，因为第二个 bash 是来自于第一个所产生的嘛！另外，每部主机的程序启动状态都不一样，所以在你的系统上面看到的 PID 与我这里的显示一定不同！那是正常的！详细的 ps 指令我们会在本章稍后介绍， 这里你只要知道 ps -l 可以查阅到相关的程序信息即可。

很多朋友常常会发现：“咦！明明我将有问题的程序关闭了，怎么过一阵子他又自动的产生？而且新产生的那个程序的 PID 与原先的还不一样，这是怎么回事呢？”不要怀疑，如果不是 crontab 工作调度的影响，肯定有一支父程序存在，所以你杀掉子程序后， 父程序就会主动再生一支！那怎么办？正所谓这：“擒贼先擒王”，找出那支父程序，然后将他删除就对啦！

- fork and exec：程序调用的流程

其实子程序与父程序之间的关系还挺复杂的，最大的复杂点在于程序互相之间的调用。在 Linux 的程序调用通常称为 fork-and-exec 的流程 [1]！程序都会借由父程序以复制 （fork） 的方式产生一个一模一样的子程序， 然后被复制出来的子程序再以 exec 的方式来执行实际要进行的程序，最终就成为一个子程序的存在。 整个流程有点像下面这张图：

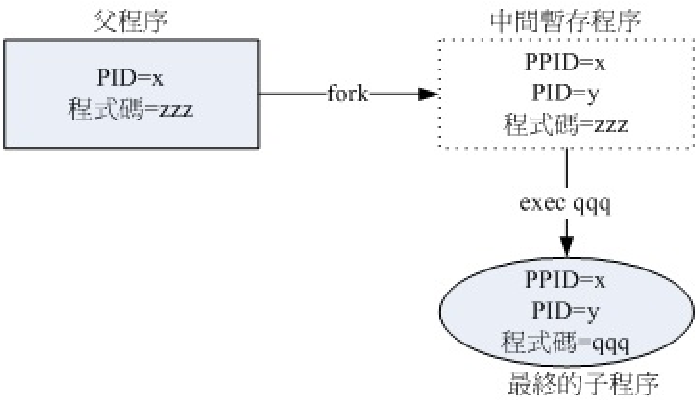


图16.1.4、程序使用 fork and exec 调用的情况示意图

（1）系统先以 fork 的方式复制一个与父程序相同的暂存程序，这个程序与父程序唯一的差别就是 PID 不同！ 但是这个暂存程序还会多一个 PPID 的参数，PPID 如前所述，就是父程序的程序识别码啦！然后（2）暂存程序开始以 exec 的方式载入实际要执行的程序，以上述图示来讲，新的程序名称为 qqq ，最终子程序的程序码就会变成 qqq 了！ 这样了解乎！

- 系统或网络服务：常驻在内存的程序

如果就我们之前学到的一些指令数据来看，其实我们下达的指令都很简单，包括用 ls 显示文件啊、用 touch 创建文件啊、rm/mkdir/cp/mv 等指令管理文件啊、chmod/chown/passwd 等等的指令来管理权限等等的，不过， 这些指令都是执行完就结束了。也就是说，该项指令被触发后所产生的 PID 很快就会终止呢！ 那有没有一直在执行的程序啊？当然有啊！而且多的是呢！

举个简单的例子来说好了，我们知道系统每分钟都会去扫瞄 /etc/crontab 以及相关的配置文件， 来进行工作调度吧？那么那个工作调度是谁负责的？当然不是鸟哥啊！ 呵呵！是 crond 这个程序所管理的，我们将他启动在背景当中一直持续不断的运行， 套句鸟哥以前 DOS 年代常常说的一句话，那就是“常驻在内存当中的程序”啦！

常驻在内存当中的程序通常都是负责一些系统所提供的功能以服务使用者各项任务，因此这些常驻程序就会被我们称为：服务 （daemon）。系统的服务非常的多， 不过主要大致分成系统本身所需要的服务，例如刚刚提到的 crond 及 atd ，还有 rsyslogd 等等的。还有一些则是负责网络连线的服务，例如 Apache, named, postfix, vsftpd... 等等的。这些网络服务比较有趣的地方，在于这些程序被执行后，他会启动一个可以负责网络监听的端口 （port） ，以提供外部用户端 （client） 的连线要求。



Tips 以 crontab 来说，他的主要执行程序名称应该是 cron 或 at 才对，为啥要加个 d 在后面？而成为 crond, atd 呢？就是因为 Linux 希望我们可以简单的判断该程序是否为 daemon，所以，一般 daemon 类型的程序都会加上 d 在文件名后头～包括服务器篇我们会看到的

httpd, vsftpd 等等都是 ^_^。

16.1.2 Linux 的多用户多任务环境

我们现在知道了，其实在 Linux 下面执行一个指令时，系统会将相关的权限、属性、程序码与数据等均载入内存， 并给予这个单元一个程序识别码 （PID），最终该指令可以进行的任务则与这个 PID 的权限有关。根据这个说明，我们就可以简单的了解，为什么 Linux 这么多用户，但是却每个人都可以拥有自己的环境了吧！^_^ ！下面我们来谈谈 Linux 多用户多任务环境的特色：

- 多人环境：

Linux 最棒的地方就在于他的多用户多任务环境了！那么什么是“多用户多任务”？在 Linux 系统上面具有多种不同的帐号， 每种帐号都有都有其特殊的权限，只有一个人具有至高无上的权力，那就是 root （系统管理员）。除了 root 之外，其他人都必须要受一些限制的！而每个人进入 Linux 的环境设置都可以随着每个人的喜好来设置 （还记得我们在第十章 BASH 提过的 ~/.bashrc 吧？对了！就是那个光！）！现在知道为什么了吧？因为每个人登陆后取得的 shell 的 PID 不同嘛！

- 多任务行为：

我们在第零章谈到 CPU 的速度，目前的 CPU 速度可高达几个 GHz。 这代表 CPU 每秒钟可以运行 109 这么多次指令。我们的 Linux 可以让 CPU 在各个工作间进行切换， 也就是说，其实每个工作都仅占去 CPU 的几个指令次数，所以 CPU 每秒就能够在各个程序之间进行切换啦！ 谁叫 CPU 可以在一秒钟进行这么多次的指令运行。

CPU 切换程序的工作，与这些工作进入到 CPU 运行的调度 （CPU 调度，非 crontab 调度）会影响到系统的整体性能！ 目前 Linux 使用的多任务切换行为是非常棒的一个机制，几乎可以将 PC 的性能整个压榨出来！ 由于性能非常好，因此当多人同时登陆系统时，其实会感受到整部主机好像就为了你存在一般！ 这就是多用户多任务的环境啦！[2]

- 多重登陆环境的七个基本终端窗口：

在 Linux 当中，默认提供了六个文字界面登陆窗口，以及一个图形界面，你可以使用 [Alt]+[F1].....[F7] 来切换不同的终端机界面，而且每个终端机界面的登陆者还可以不同人！ 很炫吧！这个东西可就很有用啦！尤其是在某个程序死掉的时候！

其实，这也是多任务环境下所产生的一个情况啦！我们的 Linux 默认会启动六个终端机登陆环境的程序，所以我们就会有六个终端机接口。 您也可以减少啊！就是减少启动的终端机程序就好了。 未来我们在开机管理流程 （第十九章） 会再仔细的介绍的！

- 特殊的程序管理行为：

以前的鸟哥笨笨的，总是以为使用 Windows 98 就可以啦！后来，因为工作的关系，需要使用 Unix 系统，想说我只要在工作机前面就好，才不要跑来跑去的到 Unix 工作站前面去呢！所以就使用 Windows 连到我的 Unix 工作站工作！好死不死，我一个程序跑下来要 2~3 天，唉～偏偏常常到了第 2.5 天的时候， Windows 98 就给他挂点去！当初真的是给他怕死了～

后来因为换了新计算机，用了随机版的 Windows 2000 ，呵呵，这东西真不错 （指对单人而言） ，在死机的时候， 他可以仅将错误的程序踢掉，而不干扰其他的程序进行，呵呵！ 从此以后，就不用担心会死机连连啰！不过，2000 毕竟还不够好，因为有的时候还是会死当！

那么 Linux 会有这样的问题吗？老实说， Linux 几乎可以说绝对不会死机的！因为他可以在任何时候， 将某个被困住的程序杀掉，然后再重新执行该程序而不用重新开机！够炫吧！那么如果我在 Linux 下以文字界面登陆，在屏幕当中显示错误讯息后就挂了～动都不能动，该如何是好！？ 这个时候那默认的七个窗口就帮上忙啦！你可以随意的再按 [Alt]+[F1].....[F7] 来切换到其他的终端机界面，然后以 ps -aux 找出刚刚的错误程序，然后给他 kill 一下，哈哈，回到刚刚的终端机界面！恩～棒！又回复正常啰！

为什么可以这样做呢？我们刚刚不是提过吗？每个程序之间可能是独立的，也可能有相依性， 只要到独立的程序当中，删除有问题的那个程序，当然他就可以被系统移除掉啦！^_^

- bash 环境下的工作管理 （job control）

我们在上一个小节有提到所谓的“父程序、子程序”的关系，那我们登陆 bash 之后， 就是取得一个名为 bash 的 PID 了，而在这个环境下面所执行的其他指令， 就几乎都是所谓的子程序了。那么，在这个单一的 bash 接口下，我可不可以进行多个工作啊？ 当然可以啦！可以“同时”进行喔！举例来说，我可以这样做：

```
[root@study ~]# cp file1 file2 &
```

在这一串指令中，重点在那个 & 的功能，他表示将 file1 这个文件复制为 file2 ，且放置于背景中执行， 也就是说执行这一个命令之后，在这一个终端接口仍然可以做其他的工作！而当这一个指令 （cp file1 file2） 执行完毕之后，系统将会在你的终端接口显示完成的消息！很便利喔！

- 多用户多任务的系统资源分配问题考虑：

多用户多任务确实有很多的好处，但其实也有管理上的困扰，因为使用者越来越多， 将导致你管理上的困扰哩！另外，由于使用者日盛，当使用者达到一定的人数后， 通常你的机器便需要升级了，因为 CPU 的运算与 RAM 的大小可能就会不敷使用！

举个例子来说，鸟哥之前的网站管理的有点不太好，因为使用了一个很复杂的人数统计程序， 这个程序会一直去取用 MySQL 数据库的数据，偏偏因为流量大，造成 MySQL 很忙碌。 在这样的情况下，当鸟哥要登陆去写网页数据，或者要去使用讨论区的资源时， 哇！慢的很！简直就是“龟速”啊！后来终于将这个程序停止不用了， 以自己写的一个小程序来取代，呵呵！这样才让 CPU 的负载 （loading） 整个降下来～ 用起来顺畅多了！ ^_^

16.2 工作管理 （job control）

这个工作管理 （job control） 是用在 bash 环境下的，也就是说：“当我们登陆系统取得 bash shell 之后，在单一终端机接口下同时进行多个工作的行为管理 ”。举例来说，我们在登陆 bash 后， 想要一边复制文件、一边进行数据搜寻、一边进行编译，还可以一边进行 vim 程序撰写！ 当然我们可以重复登陆那六个命令行的终端机环境中，不过，能不能在一个 bash 内达成？ 当然可以啊！就是使用 job control 啦！ ^_^

16.2.1 什么是工作管理？

从上面的说明当中，你应该要了解的是：“进行工作管理的行为中， 其实每个工作都是目前 bash 的子程序，亦即彼此之间是有相关性的。 我们无法以 job control 的方式由 tty1 的环境去管理 tty2 的 bash ！” 这个概念请你得先创建起来，后续的范例介绍之后，你就会清楚的了解啰！

或许你会觉得很奇怪啊，既然我可以在六个终端接口登陆，那何必使用 job control 呢？ 真是脱裤子放屁，多此一举啊！不要忘记了呢，我们可以在 /etc/security/limits.conf （第十三章） 里面设置使用者同时可以登陆的连线数，在这样的情况下，某些使用者可能仅能以一个连线来工作呢！ 所以啰，你就得要了解一下这种工作管理的模式了！此外，这个章节内容也会牵涉到很多的数据流重导向，所以，如果忘记的话， 务必回到第十章 BASH Shell 看一看喔！

由于假设我们只有一个终端接口，因此在可以出现提示字符让你操作的环境就称为前景 （foreground） ，至于其他工作就可以让你放入背景 （background） 去暂停或运行。要注意的是，放入背景的工作想要运行时， 他必须不能够与使用者互动。举例来说， vim 绝对不可能在背景里面执行 （running） 的！因为你没有输入数据他就不会跑啊！ 而且放入背景的工作是不可以使用 [ctrl]+c 来终止的！

总之，要进行 bash 的 job control 必须要注意到的限制是：

- 这些工作所触发的程序必须来自于你 shell 的子程序（只管理自己的 bash）；
- 前景：你可以控制与下达指令的这个环境称为前景的工作 （foreground）；
- 背景：可以自行运行的工作，你无法使用 [ctrl]+c 终止他，可使用 bg/fg 调用该工作；
- 背景中“执行”的程序不能等待 terminal/shell 的输入（input）

接下来让我们实际来管理这些工作吧！

16.2.2 job control 的管理

如前所述，bash 只能够管理自己的工作而不能管理其他 bash 的工作，所以即使你是 root 也不能够将别人的 bash 下面的 job 给他拿过来执行。此外，又分前景与背景，然后在背景里面的工作状态又可以分为“暂停 （stop） ”与“运行中 （running） ”。那实际进行 job 控制的指令

有哪些？下面就来谈谈。

- 直接将指令丢到背景中“执行”的 &

如同前面提到的，我们在只有一个 bash 的环境下，如果想要同时进行多个工作， 那么可以将某些工作直接丢到背景环境当中，让我们可以继续操作前景的工作！那么如何将工作丢到背景中？ 最简单的方法就是利用“ & ”这个玩意儿了！举个简单的例子，我们要将 /etc/ 整个备份成为 /tmp/etc.tar.gz 且不想要等待，那么可以这样做：

```
[root@study ~]# tar -zpcf /tmp/etc.tar.gz /etc &
[1] 14432  &lt;== [job number] PID
[root@study ~]# tar: Removing leading `/' from member names
# 在中括号内的号码为工作号码 （job number），该号码与 bash 的控制有关。
# 后续的 14432 则是这个工作在系统中的 PID。至于后续出现的数据是 tar 执行的数据流，
# 由于我们没有加上数据流重导向，所以会影响画面！不过不会影响前景的操作喔！
```

仔细的瞧一瞧，我在输入一个指令后，在该指令的最后面加上一个“ & ”代表将该指令丢到背景中， 此时 bash 会给予这个指令一个“工作号码（job number）”，就是那个 [1] 啦！至于后面那个 14432 则是该指令所触发的“ PID ”了！而且，有趣的是，我们可以继续操作 bash 呢！很不赖吧！ 不过，那么丢到背景中的工作什么时候完成？完成的时候会显示什么？如果你输入几个指令后，突然出现这个数据：

```
[1]+  Done                    tar -zpcf /tmp/etc.tar.gz /etc
```

就代表 [1] 这个工作已经完成 （Done） ，该工作的指令则是接在后面那一串命令行。 这样了解了吧！另外，这个 & 代表：“将工作丢到背景中去执行”喔！ 注意到那个“执行”的字眼！此外，这样的情况最大的好处是： 不怕被 [ctrl]+c 中断的啦！ 此外，将工作丢到背景当中要特别注意数据的流向喔！包括上面的讯息就有出现错误讯息，导致我的前景被影响。 虽然只要按下 [enter] 就会出现提示字符。但如果我将刚刚那个指令改成：

```
[root@study ~]# tar -zpcvf /tmp/etc.tar.gz /etc &
```

情况会怎样？在背景当中执行的指令，如果有 stdout 及 stderr 时，他的数据依旧是输出到屏幕上面的， 所以，我们会无法看到提示字符，当然也就无法完好的掌握前景工作。同时由于是背景工作的 tar ， 此时你怎么按下 [ctrl]+c 也无法停止屏幕被搞的花花绿绿的！所以啰，最佳的状况就是利用数据流重导向， 将输出数据传送至某个文件中。举例来说，我可以这样做：

```
[root@study ~]# tar -zpcvf /tmp/etc.tar.gz /etc &gt; /tmp/log.txt 2&gt;&1 &
[1] 14547
[root@study ~]#
```

呵呵！如此一来，输出的信息都给他传送到 /tmp/log.txt 当中，当然就不会影响到我们前景的作业了。 这样说，您应该可以更清楚数据流重导向的重要性了吧！^_^



Tips 工作号码 （job number） 只与你这个 bash 环境有关，但是他既然是个指令触发的咚咚，所以当然一定是一个程序， 因此你会观察到有 job number 也搭配一个 PID ！

- 将“目前”的工作丢到背景中“暂停”：[ctrl]-z

想个情况：如果我正在使用 vim ，却发现我有个文件不知道放在哪里，需要到 bash 环境下进行搜寻，此时是否要结束 vim 呢？呵呵！当然不需要啊！只要暂时将 vim 给他丢到背景当中等待即可。 例如以下的案例：

```
[root@study ~]# vim  ~/.bashrc
# 在 vim 的一般模式下，按下 [ctrl]-z 这两个按键
[1]+  Stopped                 vim ~/.bashrc
[root@study ~]#   &lt;==顺利取得了前景的操控权！
[root@study ~]# find / -print
....（输出省略）....
# 此时屏幕会非常的忙碌！因为屏幕上会显示所有的文件名。请按下 [ctrl]-z 暂停
[2]+  Stopped                 find / -print
```

在 vim 的一般模式下，按下 [ctrl] 及 z 这两个按键，屏幕上会出现 [1] ，表示这是第一个工作， 而那个 + 代表最近一个被丢进背景的工作，且目前在背景下默认会被取用的那个工作 （与 fg 这个指令有关 ）！而那个 Stopped 则代表目前这个工作的状态。在默认的情况下，使用 [ctrl]-z 丢到背景当中的工作都是“暂停”的状态喔！

- 观察目前的背景工作状态： jobs

```
[root@study ~]# jobs [-lrs]
选项与参数：
-l  ：除了列出 job number 与指令串之外，同时列出 PID 的号码；
-r  ：仅列出正在背景 run 的工作；
-s  ：仅列出正在背景当中暂停 （stop） 的工作。

范例一：观察目前的 bash 当中，所有的工作，与对应的 PID
[root@study ~]# jobs -l
[1]- 14566 Stopped                 vim ~/.bashrc
[2]+ 14567 Stopped                 find / -print
```

如果想要知道目前有多少的工作在背景当中，就用 jobs 这个指令吧！一般来说，直接下达 jobs 即可！ 不过，如果你还想要知道该 job number 的 PID 号码，可以加上 -l 这个参数啦！ 在输出的信息当中，例如上表，仔细看到那个 + - 号喔！那个 + 代表默认的取用工作。 所以说：“目前我有两个工作在背景当中，两个工作都是暂停的， 而如果我仅输入 fg 时，那么那个 [2] 会被拿到前景当中来处理”！

其实 + 代表最近被放到背景的工作号码， - 代表最近最后第二个被放置到背景中的工作号码。 而超过最后第三个以后的工作，就不会有 +/- 符号存在了！

- 将背景工作拿到前景来处理：fg

刚刚提到的都是将工作丢到背景当中去执行的，那么有没有可以将背景工作拿到前景来处理的？ 有啊！就是那个 fg （foreground） 啦！举例来说，我们想要将上头范例当中的工作拿出来处理时：

```
[root@study ~]# fg %jobnumber
选项与参数：
%jobnumber ：jobnumber 为工作号码（数字）。注意，那个 % 是可有可无的！

范例一：先以 jobs 观察工作，再将工作取出：
[root@study ~]# jobs -l
[1]- 14566 Stopped                 vim ~/.bashrc
[2]+ 14567 Stopped                 find / -print
[root@study ~]# fg      &lt;==默认取出那个 + 的工作，亦即 [2]。立即按下[ctrl]-z
[root@study ~]# fg %1   &lt;==直接规定取出的那个工作号码！再按下[ctrl]-z
[root@study ~]# jobs -l
[1]+ 14566 Stopped                 vim ~/.bashrc
[2]- 14567 Stopped                 find / -print
```

经过 fg 指令就能够将背景工作拿到前景来处理啰！不过比较有趣的是最后一个显示的结果，我们会发现 + 出现在第一个工作后！ 怎么会这样啊？这是因为你刚刚利用 fg %1 将第一号工作捉到前景后又放回背景，此时最后一个被放入背景的将变成 vi 那个指令动作， 所以当然 [1] 后面就会出现 + 了！了解乎！另外，如果输入“ fg - ” 则代表将 - 号的那个工作号码拿出来，上面就是 [2]- 那个工作号码啦！

- 让工作在背景下的状态变成运行中： bg

我们刚刚提到，那个 [ctrl]-z 可以将目前的工作丢到背景下面去“暂停”， 那么如何让一个工作在背景下面“ Run ”呢？我们可以在下面这个案例当中来测试！ 注意喔！下面的测试要进行的快一点！^_^

```
范例一：一执行 find / -perm /7000 &gt; /tmp/text.txt 后，立刻丢到背景去暂停！
[root@study ~]# find / -perm /7000 &gt; /tmp/text.txt
# 此时，请立刻按下 [ctrl]-z 暂停！
[3]+  Stopped                 find / -perm /7000 &gt; /tmp/text.txt

范例二：让该工作在背景下进行，并且观察他！！
[root@study ~]# jobs ; bg %3 ; jobs
[1]   Stopped                 vim ~/.bashrc
[2]-  Stopped                 find / -print
[3]+  Stopped                 find / -perm /7000 &gt; /tmp/text.txt
[3]+ find / -perm /7000 &gt; /tmp/text.txt &
[1]-  Stopped                 vim ~/.bashrc
[2]+  Stopped                 find / -print
[3]   Running                 find / -perm /7000 &gt; /tmp/text.txt &
```

看到哪里有差异吗？呼呼！没错！就是那个状态列～以经由 Stopping 变成了 Running 啰！看到差异点，嘿嘿！命令行最后方多了一个 & 的符号啰！ 代表该工作被启动在背景当中了啦！ ^_^

- 管理背景当中的工作： kill

刚刚我们可以让一个已经在背景当中的工作继续工作，也可以让该工作以 fg 拿到前景来， 那么，如果想要将该工作直接移除呢？或者是将该工作重新启动呢？这个时候就得需要给予该工作一个讯号 （signal） ，让他知道该怎么作才好啊！此时， kill 这个指令就派上用场啦！

```
[root@study ~]# kill -signal %jobnumber
[root@study ~]# kill -l
选项与参数：
-l   ：这个是 L 的小写，列出目前 kill 能够使用的讯号 （signal） 有哪些？
signal ：代表给予后面接的那个工作什么样的指示啰！用 man 7 signal 可知：
  -1 ：重新读取一次参数的配置文件 （类似 reload）；
  -2 ：代表与由键盘输入 [ctrl]-c 同样的动作；
  -9 ：立刻强制删除一个工作；
  -15：以正常的程序方式终止一项工作。与 -9 是不一样的。

范例一：找出目前的 bash 环境下的背景工作，并将该工作“强制删除”。
[root@study ~]# jobs
[1]+  Stopped                 vim ~/.bashrc
[2]   Stopped                 find / -print
[root@study ~]# kill -9 %2; jobs
[1]+  Stopped                 vim ~/.bashrc
[2]   Killed                  find / -print
# 再过几秒你再下达 jobs 一次，就会发现 2 号工作不见了！因为被移除了！

范例二：找出目前的 bash 环境下的背景工作，并将该工作“正常终止”掉。
[root@study ~]# jobs
[1]+  Stopped                 vim ~/.bashrc
[root@study ~]# kill -SIGTERM %1
# -SIGTERM 与 -15 是一样的！您可以使用 kill -l 来查阅！
# 不过在这个案例中， vim 的工作无法被结束喔！因为他无法通过 kill 正常终止的意思！
```

特别留意一下， -9 这个 signal 通常是用在“强制删除一个不正常的工作”时所使用的， -15 则是以正常步骤结束一项工作（15也是默认值），两者之间并不相同呦！举上面的例子来说，我用 vim 的时候，不是会产生一个 .filename.swp 的文件吗？ 那么，当使用 -15 这个 signal 时， vim 会尝试以正常的步骤来结束掉该 vi 的工作， 所以 .filename.swp 会主动的被移除。但若是使用 -9 这个 signal 时，由于该 vim 工作会被强制移除掉，因此， .filename.swp 就会继续存在文件系统当中。这样您应该可以稍微分辨一下了吧？

不过，毕竟正常的作法中，你应该先使用 fg 来取回前景控制权，然后再离开 vim 才对～因此，以上面的范例二为例，其实 kill 确实无法使用 -15 正常的结束掉 vim 的动作喔！此时还是不建议使用 -9 啦！因为你知道如何正常结束该程序不是吗？ 通常使用 -9 是因为某些程序你真的不知道怎么通过正常手段去终止他，这才用到 -9 的！

其实， kill 的妙用是很无穷的啦！他搭配 signal 所详列的信息 （用 man 7 signal 去查阅相关数据） 可以让您有效的管理工作与程序 （Process） ，此外，那个 killall 也是同样的用法！ 至于常用的 signal 您至少需要了解 1, 9, 15 这三个 signal 的意义才好。 此外， signal 除了以数值来表示之外，也可以使用讯号名称喔！ 举例来说，上面的范例二就是一个例子啦！至于 signal number 与名称的对应， 呵呵，使用 kill -l 就知道啦（L的小写）！

另外， kill 后面接的数字默认会是 PID ，如果想要管理 bash 的工作控制，就得要加上 %数字 了， 这点也得特别留意才行喔！

16.2.3 离线管理问题

要注意的是，我们在工作管理当中提到的“背景”指的是在终端机模式下可以避免 [crtl]-c 中断的一个情境， 你可以说那个是 bash 的背景，并不是放到系统的背景去喔！所以，工作管理的背景依旧与终端机有关啦！ 在这样的情况下，如果你是以远端连线方式连接到你的 Linux 主机，并且将工作以 & 的方式放到背景去， 请问，在工作尚未结束的情况下你离线了，该工作还会继续进行吗？答案是“否”！不会继续进行，而是会被中断掉。

那怎么办？如果我的工作需要进行一大段时间，我又不能放置在背景下面，那该如何处理呢？ 首先，你可以参考前一章的 at 来处理即可！因为 at 是将工作放置到系统背景， 而与终端机无关。如果不想要使用 at 的话，那你也可以尝试使用 nohup 这个指令来处理喔！这个 nohup 可以让你在离线或登出系统后，还能够让工作继续进行。他的语法有点像这样：

```
[root@study ~]# nohup [指令与参数]   &lt;==在终端机前景中工作
[root@study ~]# nohup [指令与参数] & &lt;==在终端机背景中工作
```

有够好简单的指令吧！上述指令需要注意的是， nohup 并不支持 bash 内置的指令，因此你的指令必须要是外部指令才行。 我们来尝试玩一下下面的任务吧！

```
# 1\. 先编辑一支会“睡着 500 秒”的程序：
[root@study ~]# vim sleep500.sh
#!/bin/bash
/bin/sleep 500s
/bin/echo "I have slept 500 seconds."

# 2\. 丢到背景中去执行，并且立刻登出系统：
[root@study ~]# chmod a+x sleep500.sh
[root@study ~]# nohup ./sleep500.sh &
[2] 14812
[root@study ~]#  nohup: ignoring input and appending output to `nohup.out' &lt;==会告知这个
[root@study ~]# exit
```

如果你再次登陆的话，再使用 pstree 去查阅你的程序，会发现 sleep500.sh 还在执行中喔！并不会被中断掉！ 这样了解意思了吗？由于我们的程序最后会输出一个讯息，但是 nohup 与终端机其实无关了， 因此这个讯息的输出就会被导向“ ~/nohup.out ”，所以你才会看到上述指令中，当你输入 nohup 后， 会出现那个提示讯息啰。

如果你想要让在背景的工作在你登出后还能够继续的执行，那么使用 nohup 搭配 & 是不错的运行情境喔！ 可以参考看看！

16.3 程序管理

本章一开始就提到所谓的“程序”的概念，包括程序的触发、子程序与父程序的相关性等等，此外，还有那个“程序的相依性”以及所谓的“僵尸程序”等等需要说明的呢！为什么程序管理这么重要呢？这是因为：

- 首先，本章一开始就谈到的，我们在操作系统时的各项工作其实都是经过某个 PID 来达成的 （包括你的 bash 环境）， 因此，能不能进行某项工作，就与该程序的权限有关了。
- 再来，如果您的 Linux 系统是个很忙碌的系统，那么当整个系统资源快要被使用光时，您是否能够找出最耗系统的那个程序，然后删除该程序，让系统恢复正常呢？
- 此外，如果由于某个程序写的不好，导致产生一个有问题的程序在内存当中，您又该如何找出他，然后将他移除呢？
- 如果同时有五六项工作在您的系统当中运行，但其中有一项工作才是最重要的， 该如何让那一项重要的工作被最优先执行呢？

所以啰，一个称职的系统管理员，必须要熟悉程序的管理流程才行，否则当系统发生问题时，还真是很难解决问题呢！ 下面我们会先介绍如何观察程序与程序的状态，然后再加以程序控制啰！

16.3.1 程序的观察

既然程序这么重要，那么我们如何查阅系统上面正在运行当中的程序呢？很简单啊！ 利用静态的 ps 或者是动态的 top，还能以 pstree 来查阅程序树之间的关系喔！

- ps ：将某个时间点的程序运行情况撷取下来

```
[root@study ~]# ps aux  &lt;==观察系统所有的程序数据
[root@study ~]# ps -lA  &lt;==也是能够观察所有系统的数据
[root@study ~]# ps axjf &lt;==连同部分程序树状态
选项与参数：
-A  ：所有的 process 均显示出来，与 -e 具有同样的效用；
-a  ：不与 terminal 有关的所有 process ；
-u  ：有效使用者 （effective user） 相关的 process ；
x   ：通常与 a 这个参数一起使用，可列出较完整信息。
输出格式规划：
l   ：较长、较详细的将该 PID 的的信息列出；
j   ：工作的格式 （jobs format）
-f  ：做一个更为完整的输出。
```

鸟哥个人认为 ps 这个指令的 man page 不是很好查阅，因为很多不同的 Unix 都使用这个 ps 来查阅程序状态， 为了要符合不同版本的需求，所以这个 man page 写的非常的庞大！因此，通常鸟哥都会建议你，直接背两个比较不同的选项， 一个是只能查阅自己 bash 程序的“ ps -l ”一个则是可以查阅所有系统运行的程序“ ps aux ”！注意，你没看错，是“ ps aux ”没有那个减号（-）！先来看看关于自己 bash 程序状态的观察：

- 仅观察自己的 bash 相关程序： ps -l

```
范例一：将目前属于您自己这次登陆的 PID 与相关信息列示出来（只与自己的 bash 有关）
[root@study ~]# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 14830 13970  0  80   0 - 52686 poll_s pts/0    00:00:00 sudo
4 S     0 14835 14830  0  80   0 - 50511 wait   pts/0    00:00:00 su
4 S     0 14836 14835  0  80   0 - 29035 wait   pts/0    00:00:00 bash
0 R     0 15011 14836  0  80   0 - 30319 -      pts/0    00:00:00 ps
# 还记得鸟哥说过，非必要不要使用 root 直接登陆吧？从这个 ps -l 的分析，你也可以发现，
# 鸟哥其实是使用 sudo 才转成 root 的身份～否则连测试机，鸟哥都是使用一般帐号登陆的！
```

系统整体的程序运行是非常多的，但如果使用 ps -l 则仅列出与你的操作环境（bash）有关的程序而已，亦即最上层的父程序会是你自己的 bash 而没有延伸到 systemd（后续会交待！）这支程序去！那么 ps -l 秀出来的数据有哪些呢？我们就来观察看看：

- F：代表这个程序旗标（process flags），说明这个程序的总结权限，常见号码有：
    - 若为 4 表示此程序的权限为 root ；
    - 若为 1 则表示此子程序仅进行复制（fork）而没有实际执行（exec）。
- S：代表这个程序的状态（STAT），主要的状态有：
    - R（Running）：该程序正在运行中；
    - S（Sleep）：该程序目前正在睡眠状态（idle），但可以被唤醒（signal）。
    - D ：不可被唤醒的睡眠状态，通常这支程序可能在等待 I/O 的情况（ex>打印）
    - T ：停止状态（stop），可能是在工作控制（背景暂停）或除错（traced）状态；
    - Z（Zombie）：僵尸状态，程序已经终止但却无法被移除至内存外。
- UID/PID/PPID：代表“此程序被该 UID 所拥有/程序的 PID 号码/此程序的父程序 PID 号码”
- C：代表 CPU 使用率，单位为百分比；
- PRI/NI：Priority/Nice 的缩写，代表此程序被 CPU 所执行的优先顺序，数值越小代表该程序越快被 CPU 执行。详细的 PRI 与 NI 将在下一小节说明。
- ADDR/SZ/WCHAN：都与内存有关，ADDR 是 kernel function，指出该程序在内存的哪个部分，如果是个 running 的程序，一般就会显示“ - ” / SZ 代表此程序用掉多少内存 / WCHAN 表示目前程序是否运行中，同样的，若为 - 表示正在运行中。
- TTY：登陆者的终端机位置，若为远端登陆则使用动态终端接口（pts/n）；
- TIME：使用掉的 CPU 时间，注意，是此程序实际花费 CPU 运行的时间，而不是系统时间；
- CMD：就是 command 的缩写，造成此程序的触发程序之指令为何。

所以你看到的 ps -l 输出讯息中，他说明的是：“bash 的程序属于 UID 为 0 的使用者，状态为睡眠（sleep），之所以为睡眠因为他触发了 ps（状态为 run）之故。此程序的 PID 为 14836，优先执行顺序为 80 ，下达 bash 所取得的终端接口为 pts/0 ，运行状态为等待

（wait） 。”这样已经够清楚了吧？ 您自己尝试解析一下那么 ps 那一行代表的意义为何呢？ ^_^

接下来让我们使用 ps 来观察一下系统内所有的程序状态吧！

- 观察系统所有程序： ps aux

```
范例二：列出目前所有的正在内存当中的程序：
[root@study ~]# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2  60636  7948 ?        Ss   Aug04   0:01 /usr/lib/systemd/systemd
root         2  0.0  0.0      0     0 ?        S    Aug04   0:00 [kthreadd]
.....（中间省略）.....
root     14830  0.0  0.1 210744  3988 pts/0    S    Aug04   0:00 sudo su -
root     14835  0.0  0.1 202044  2996 pts/0    S    Aug04   0:00 su -
root     14836  0.0  0.1 116140  2960 pts/0    S    Aug04   0:00 -bash
.....（中间省略）.....
root     18459  0.0  0.0 123372  1380 pts/0    R+   00:25   0:00 ps aux
```

你会发现 ps -l 与 ps aux 显示的项目并不相同！在 ps aux 显示的项目中，各字段的意义为：

- USER：该 process 属于那个使用者帐号的？
- PID ：该 process 的程序识别码。
- %CPU：该 process 使用掉的 CPU 资源百分比；
- %MEM：该 process 所占用的实体内存百分比；
- VSZ ：该 process 使用掉的虚拟内存量 （KBytes）
- RSS ：该 process 占用的固定的内存量 （KBytes）
- TTY ：该 process 是在那个终端机上面运行，若与终端机无关则显示 ?，另外， tty1-tty6 是本机上面的登陆者程序，若为 pts/0 等等的，则表示为由网络连接进主机的程序。
- STAT：该程序目前的状态，状态显示与 ps -l 的 S 旗标相同 （R/S/T/Z）
- START：该 process 被触发启动的时间；
- TIME ：该 process 实际使用 CPU 运行的时间。
- COMMAND：该程序的实际指令为何？

一般来说，ps aux 会依照 PID 的顺序来排序显示，我们还是以 14836 那个 PID 那行来说明！该行的意义为“ root 执行的 bash PID 为 14836，占用了 0.1% 的内存容量百分比，状态为休眠 （S） ，该程序启动的时间为 8 月 4 号，因此启动太久了， 所以没有列出实际的时间点。且取得的终端机环境为 pts/0 。”与 ps aux 看到的其实是同一个程序啦！这样可以理解吗？ 让我们继续使用 ps 来观察一下其他的信息吧！

```
范例三：以范例一的显示内容，显示出所有的程序：
[root@study ~]# ps -lA
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  80   0 - 15159 ep_pol ?        00:00:01 systemd
1 S     0     2     0  0  80   0 -     0 kthrea ?        00:00:00 kthreadd
1 S     0     3     2  0  80   0 -     0 smpboo ?        00:00:00 ksoftirqd/0
....（以下省略）....
# 你会发现每个字段与 ps -l 的输出情况相同，但显示的程序则包括系统所有的程序。

范例四：列出类似程序树的程序显示：
[root@study ~]# ps axjf
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    0     2     0     0 ?           -1 S        0   0:00 [kthreadd]
    2     3     0     0 ?           -1 S        0   0:00  \_ [ksoftirqd/0]
.....（中间省略）.....
    1  1326  1326  1326 ?           -1 Ss       0   0:00 /usr/sbin/sshd -D
 1326 13923 13923 13923 ?           -1 Ss       0   0:00  \_ sshd: dmtsai [priv]
13923 13927 13923 13923 ?           -1 S     1000   0:00      \_ sshd: dmtsai@pts/0
13927 13928 13928 13928 pts/0    18703 Ss    1000   0:00          \_ -bash
13928 13970 13970 13928 pts/0    18703 S     1000   0:00              \_ bash
13970 14830 14830 13928 pts/0    18703 S        0   0:00                  \_ sudo su -
14830 14835 14830 13928 pts/0    18703 S        0   0:00                      \_ su -
14835 14836 14836 13928 pts/0    18703 S        0   0:00                          \_ -bas
14836 18703 18703 13928 pts/0    18703 R+       0   0:00                              \_
.....（后面省略）.....
```

看出来了吧？其实鸟哥在进行一些测试时，都是以网络连线进虚拟机来测试的，所以啰，你会发现其实程序之间是有相关性的啦！ 不过，其实还可以使用 pstree 来达成这个程序树喔！ 以上面的例子来看，鸟哥是通过 sshd 提供的网络服务取得一个程序， 该程序提供 bash 给我使用，而我通过 bash 再去执行 ps axjf ！这样可以看的懂了吗？其他各字段的意义请 man ps （虽然真的很难 man 的出来！） 啰！

```
范例五：找出与 cron 与 rsyslog 这两个服务有关的 PID 号码？
[root@study ~]# ps aux &#124; egrep '（cron&#124;rsyslog）'
root       742  0.0  0.1 208012  4088 ?        Ssl  Aug04   0:00 /usr/sbin/rsyslogd -n
root      1338  0.0  0.0 126304  1704 ?        Ss   Aug04   0:00 /usr/sbin/crond -n
root     18740  0.0  0.0 112644   980 pts/0    S+   00:49   0:00 grep -E --color=auto （c
# 所以号码是 742 及 1338 这两个啰！就是这样找的啦！
```

除此之外，我们必须要知道的是“僵尸 （zombie） ”程序是什么？ 通常，造成僵尸程序的成因是因为该程序应该已经执行完毕，或者是因故应该要终止了， 但是该程序的父程序却无法完整的将该程序结束掉，而造成那个程序一直存在内存当中。 如果你发现在某个程序的 CMD 后面还接上 <defunct> 时，就代表该程序是僵尸程序啦，例如：

```
apache  8683  0.0  0.9 83384 9992 ?   Z  14:33   0:00 /usr/sbin/httpd &lt;defunct&gt;
```

当系统不稳定的时候就容易造成所谓的僵尸程序，可能是因为程序写的不好啦，或者是使用者的操作习惯不良等等所造成。 如果你发现系统中很多僵尸程序时，记得啊！要找出该程序的父程序，然后好好的做个追踪，好好的进行主机的环境最优化啊！ 看看有什么地方需要改善的，不要只是直接将他 kill 掉而已呢！不然的话，万一他一直产生，那可就麻烦了！ @_@

事实上，通常僵尸程序都已经无法控管，而直接是交给 systemd 这支程序来负责了，偏偏 systemd 是系统第一支执行的程序， 他是所有程序的父程序！我们无法杀掉该程序的 （杀掉他，系统就死掉了！），所以啰，如果产生僵尸程序， 而系统过一阵子还没有办法通过核心非经常性的特殊处理来将该程序删除时，那你只好通过 reboot 的方式来将该程序抹去了！

- top：动态观察程序的变化

相对于 ps 是撷取一个时间点的程序状态， top 则可以持续侦测程序运行的状态！使用方式如下：

```
[root@study ~]# top [-d 数字] &#124; top [-bnp]
选项与参数：
-d  ：后面可以接秒数，就是整个程序画面更新的秒数。默认是 5 秒；
-b  ：以批次的方式执行 top ，还有更多的参数可以使用喔！
      通常会搭配数据流重导向来将批次的结果输出成为文件。
-n  ：与 -b 搭配，意义是，需要进行几次 top 的输出结果。
-p  ：指定某些个 PID 来进行观察监测而已。
在 top 执行过程当中可以使用的按键指令：
    ? ：显示在 top 当中可以输入的按键指令；
    P ：以 CPU 的使用资源排序显示；
    M ：以 Memory 的使用资源排序显示；
    N ：以 PID 来排序喔！
    T ：由该 Process 使用的 CPU 时间累积 （TIME+） 排序。
    k ：给予某个 PID 一个讯号  （signal）
    r ：给予某个 PID 重新制订一个 nice 值。
    q ：离开 top 软件的按键。
```

其实 top 的功能非常多！可以用的按键也非常的多！可以参考 man top 的内部说明文档！ 鸟哥这里仅是列出一些鸟哥自己常用的选项而已。接下来让我们实际观察一下如何使用 top 与 top 的画面吧！

```
范例一：每两秒钟更新一次 top ，观察整体信息：
[root@study ~]# top -d 2
top - 00:53:59 up  6:07,  3 users,  load average: 0.00, 0.01, 0.05
Tasks: 179 total,   2 running, 177 sleeping,   0 stopped,   0 zombie
%Cpu（s）:  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2916388 total,  1839140 free,   353712 used,   723536 buff/cache
KiB Swap:  1048572 total,  1048572 free,        0 used.  2318680 avail Mem
    &lt;==如果加入 k 或 r 时，就会有相关的字样出现在这里喔！
  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
18804 root      20   0  130028   1872   1276 R   0.5  0.1   0:00.02 top
    1 root      20   0   60636   7948   2656 S   0.0  0.3   0:01.70 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.01 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.00 ksoftirqd/0
```

top 也是个挺不错的程序观察工具！但不同于 ps 是静态的结果输出， top 这个程序可以持续的监测整个系统的程序工作状态。 在默认的情况下，每次更新程序资源的时间为 5 秒，不过，可以使用 -d 来进行修改。 top 主要分为两个画面，上面的画面为整个系统的资源使用状态，基本上总共有六行，显示的内容依序是：

- 第一行（top...）：这一行显示的信息分别为：
    - 目前的时间，亦即是 00:53:59 那个项目；
    - 开机到目前为止所经过的时间，亦即是 up 6:07, 那个项目；

  - 已经登陆系统的使用者人数，亦即是 3 users, 项目；
  - 系统在 1, 5, 15 分钟的平均工作负载。我们在第十五章谈到的 batch 工作方式为负载小于 0.8 就是这个负载啰！代表的是 1, 5, 15 分钟，系统平均要负责运行几个程序（工作）的意思。 越小代表系统越闲置，若高于 1 得要注意你的系统程序是否太过繁复了！
- 第二行（Tasks...）：显示的是目前程序的总量与个别程序在什么状态（running, sleeping, stopped, zombie）。 比较需要注意的是最后的 zombie 那个数值，如果不是 0 ！好好看看到底是那个 process 变成僵尸了吧？

- 第三行（%Cpus...）：显示的是 CPU 的整体负载，每个项目可使用 ? 查阅。需要特别注意的是 wa 项目，那个项目代表的是 I/O wait， 通常你的系统会变慢都是 I/O 产生的问题比较大！因此这里得要注意这个项目耗用 CPU 的资源喔！ 另外，如果是多核心的设备，可以按下数字键“1”来切换成不同 CPU 的负载率。

- 第四行与第五行：表示目前的实体内存与虚拟内存 （Mem/Swap） 的使用情况。 再次重申，要注意的是 swap 的使用量要尽量的少！如果 swap 被用的很大量，表示系统的实体内存实在不足！

- 第六行：这个是当在 top 程序当中输入指令时，显示状态的地方。

至于 top 下半部分的画面，则是每个 process 使用的资源情况。比较需要注意的是：

- PID ：每个 process 的 ID 啦！
- USER：该 process 所属的使用者；
- PR ：Priority 的简写，程序的优先执行顺序，越小越早被执行；
- NI ：Nice 的简写，与 Priority 有关，也是越小越早被执行；
- %CPU：CPU 的使用率；
- %MEM：内存的使用率；
- TIME+：CPU 使用时间的累加；

top 默认使用 CPU 使用率 （%CPU） 作为排序的重点，如果你想要使用内存使用率排序，则可以按下“M”， 若要回复则按下“P”即可。如果想要离开 top 则按下“ q ”吧！如果你想要将 top 的结果输出成为文件时， 可以这样做：

```
范例二：将 top 的信息进行 2 次，然后将结果输出到 /tmp/top.txt
[root@study ~]# top -b -n 2 &gt; /tmp/top.txt
# 这样一来，嘿嘿！就可以将 top 的信息存到 /tmp/top.txt 文件中了。
```

这玩意儿很有趣！可以帮助你将某个时段 top 观察到的结果存成文件，可以用在你想要在系统背景下面执行。 由于是背景下面执行，与终端机的屏幕大小无关，因此可以得到全部的程序画面！那如果你想要观察的程序 CPU 与内存使用率都很低，结果老是无法在第一行显示时，该怎办？我们可以仅观察单一程序喔！如下所示：

```
范例三：我们自己的 bash PID 可由 $$ 变量取得，请使用 top 持续观察该 PID
[root@study ~]# echo $$
14836  &lt;==就是这个数字！他是我们 bash 的 PID
[root@study ~]# top -d 2 -p 14836
top - 01:00:53 up  6:14,  3 users,  load average: 0.00, 0.01, 0.05
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.1 sy,  0.0 ni, 99.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2916388 total,  1839264 free,   353424 used,   723700 buff/cache
KiB Swap:  1048572 total,  1048572 free,        0 used.  2318848 avail Mem

 PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
14836 root      20   0  116272   3136   1848 S   0.0  0.1   0:00.07 bash
```

看到没！就只会有一支程序给你看！很容易观察吧！好，那么如果我想要在 top 下面进行一些动作呢？ 比方说，修改 NI 这个数值呢？可以这样做：

```
范例四：承上题，上面的 NI 值是 0 ，想要改成 10 的话？
# 在范例三的 top 画面当中直接按下 r 之后，会出现如下的图样！
top - 01:02:01 up  6:15,  3 users,  load average: 0.00, 0.01, 0.05
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.1 us,  0.0 sy,  0.0 ni, 99.9 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2916388 total,  1839140 free,   353576 used,   723672 buff/cache
KiB Swap:  1048572 total,  1048572 free,        0 used.  2318724 avail Mem
PID to renice [default pid = 14836] 14836
 PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
14836 root      20   0  116272   3136   1848 S   0.0  0.1   0:00.07 bash
```

在你完成上面的动作后，在状态列会出现如下的信息：

```
Renice PID 14836 to value 10   &lt;==这是 nice 值
 PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
```

接下来你就会看到如下的显示画面！

```
top - 01:04:13 up  6:17,  3 users,  load average: 0.00, 0.01, 0.05
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us,  0.0 sy,  0.0 ni,100.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  2916388 total,  1838676 free,   354020 used,   723692 buff/cache
KiB Swap:  1048572 total,  1048572 free,        0 used.  2318256 avail Mem

 PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
14836 root      30  10  116272   3136   1848 S   0.0  0.1   0:00.07 bash
```

看到不同处了吧？底线的地方就是修改了之后所产生的效果！一般来说，如果鸟哥想要找出最损耗 CPU 资源的那个程序时，大多使用的就是 top 这支程序啦！然后强制以 CPU 使用资源来排序 （在 top 当中按下 P 即可）， 就可以很快的知道啦！ ^_^。多多爱用这个好用的东西喔！

- pstree

```
[root@study ~]# pstree [-A&#124;U] [-up]
选项与参数：
-A  ：各程序树之间的连接以 ASCII 字符来连接；
-U  ：各程序树之间的连接以万国码的字符来连接。在某些终端接口下可能会有错误；
-p  ：并同时列出每个 process 的 PID；
-u  ：并同时列出每个 process 的所属帐号名称。

范例一：列出目前系统上面所有的程序树的相关性：
[root@study ~]# pstree -A
systemd-+-ModemManager---2*[{ModemManager}]       # 这行是 ModenManager 与其子程序
        &#124;-NetworkManager---3*[{NetworkManager}]   # 前面有数字，代表子程序的数量！
....（中间省略）....
        &#124;-sshd---sshd---sshd---bash---bash---sudo---su---bash---pstree &lt;==我们指令
....（下面省略）....
# 注意一下，为了节省版面，所以鸟哥已经删去很多程序了！

范例二：承上题，同时秀出 PID 与 users
[root@study ~]# pstree -Aup
systemd（1）-+-ModemManager（745）-+-{ModemManager}（785）
           &#124;                   `-{ModemManager}（790）
           &#124;-NetworkManager（870）-+-{NetworkManager}（907）
           &#124;                     &#124;-{NetworkManager}（911）
           &#124;                     `-{NetworkManager}（914）
....（中间省略）....
           &#124;-sshd（1326）---sshd（13923）---sshd（13927,dmtsai）---bash（13928）---bash
....（下面省略）....
# 在括号 （） 内的即是 PID 以及该程序的 owner 喔！一般来说，如果该程序的所有人与父程序同，
# 就不会列出，但是如果与父程序不一样，那就会列出该程序的拥有者！看上面 13927 就转变成 dmtsai 了
```

如果要找程序之间的相关性，这个 pstree 真是好用到不行！直接输入 pstree 可以查到程序相关性，如上表所示，还会使用线段将相关性程序链接起来哩！ 一般链接符号可以使用 ASCII 码即可，但有时因为语系问题会主动的以 Unicode 的符号来链接， 但因为可能终端机无法支持该编码，或许会造成乱码问题。因此可以加上 -A 选项来克服此类线段乱码问题。

由 pstree 的输出我们也可以很清楚的知道，所有的程序都是依附在 systemd 这支程序下面的！ 仔细看一下，这支程序的 PID 是一号喔！因为他是由 Linux 核心所主动调用的第一支程序！所以 PID 就是一号了。 这也是我们刚刚提到僵尸程序时有提到，为啥发生僵尸程序需要重新开机？ 因为 systemd 要重新启动，而重新启动 systemd 就是 reboot 啰！

如果还想要知道 PID 与所属使用者，加上 -u 及 -p 两个参数即可。我们前面不是一直提到，如果子程序挂点或者是老是砍不掉子程序时，该如何找到父程序吗？呵呵！用这个 pstree 就对了！ ^_^

16.3.2 程序的管理

程序之间是可以互相控制的！举例来说，你可以关闭、重新启动服务器软件，服务器软件本身是个程序， 你既然可以让她关闭或启动，当然就是可以控制该程序啦！那么程序是如何互相管理的呢？其实是通过给予该程序一个讯号 （signal） 去告知该程序你想要让她作什么！因此这个讯号就很重要啦！

我们也在本章之前的 bash 工作管理当中提到过， 要给予某个已经存在背景中的工作某些动作时，是直接给予一个讯号给该工作号码即可。那么到底有多少 signal 呢？ 你可以使用 kill -l （小写的 L ） 或者是 man 7 signal 都可以查询到！主要的讯号代号与名称对应及内容是：

代号	名称	内容
1	SIGHUP	启动被终止的程序，可让该 PID 重新读取自己的配置文件，类似重新启动
2	SIGINT	相当于用键盘输入 [ctrl]-c 来中断一个程序的进行
9	SIGKILL	代表强制中断一个程序的进行，如果该程序进行到一半， 那么尚未完成的部分可能会有“半产品”产生，类似 vim会有 .filename.swp 保留下来。
15	SIGTERM	以正常的结束程序来终止该程序。由于是正常的终止， 所以后续的动作会将他完成。不过，如果该程序已经发生问题，就是无法使用正常的方法终止时， 输入这个 signal 也是没有用的。
19	SIGSTOP	相当于用键盘输入 [ctrl]-z 来暂停一个程序的进行

上面仅是常见的 signal 而已，更多的讯号信息请自行 man 7 signal 吧！一般来说，你只要记得“1, 9, 15”这三个号码的意义即可。那么我们如何传送一个讯号给某个程序呢？就通过 kill 或 killall 吧！下面分别来看看：

- kill -signal PID

kill 可以帮我们将这个 signal 传送给某个工作 （%jobnumber） 或者是某个 PID （直接输入数字）。 要再次强调的是： kill 后面直接加数字与加上 %number 的情况是不同的！ 这个很重要喔！因为工作控制中有 1 号工作，但是 PID 1 号则是专指“ systemd ”这支程序！你怎么可以将 systemd 关闭呢？ 关闭 systemd ，你的系统就当掉了啊！所以记得那个 % 是专门用在工作控制的喔！ 我们就活用一下 kill 与刚刚上面提到的 ps 来做个简单的练习吧！

例题：以 ps 找出 rsyslogd 这个程序的 PID 后，再使用 kill 传送讯息，使得 rsyslogd 可以重新读取配置文件。答：由于需要重新读取配置文件，因此 signal 是 1 号。至于找出 rsyslogd 的 PID 可以是这样做：

> ps aux | grep 'rsyslogd' | grep -v 'grep'| awk '{print $2}'

接下来则是实际使用 kill -1 PID，因此，整串指令会是这样：

> kill -SIGHUP $（ps aux | grep 'rsyslogd' | grep -v 'grep'| awk '{print $2}'）

如果要确认有没有重新启动 syslog ，可以参考登录文件的内容，使用如下指令查阅：

> tail -5 /var/log/messages

如果你有看到类似“Aug 5 01:25:02 study rsyslogd: [origin software="rsyslogd" swVersion="7.4.7" x-pid="742" x-info="http://www.rsyslog.com"] rsyslogd was HUPed”之类的字样，就是表示 rsyslogd 在 8/5 有重新启动 （restart） 过了！

了解了这个用法以后，如果未来你想要将某个莫名其妙的登陆者的连线删除的话，就可以通过使用 pstree -p 找到相关程序， 然后再以 kill -9 将该程序删除，该条连线就会被踢掉了！这样很简单吧！

- killall -signal 指令名称

由于 kill 后面必须要加上 PID （或者是 job number），所以，通常 kill 都会配合 ps, pstree 等指令，因为我们必须要找到相对应的那个程序的 ID 嘛！但是，如此一来，很麻烦～有没有可以利用“下达指令的名称”来给予讯号的？举例来说，能不能直接将 rsyslogd 这个程序给予一个 SIGHUP 的讯号呢？可以的！用 killall 吧！

```
[root@study ~]# killall [-iIe] [command name]
选项与参数：
-i  ：interactive 的意思，互动式的，若需要删除时，会出现提示字符给使用者；
-e  ：exact 的意思，表示“后面接的 command name 要一致”，但整个完整的指令
      不能超过 15 个字符。
-I  ：指令名称（可能含参数）忽略大小写。

范例一：给予 rsyslogd 这个指令启动的 PID 一个 SIGHUP 的讯号
[root@study ~]# killall -1 rsyslogd
# 如果用 ps aux 仔细看一下，若包含所有参数，则 /usr/sbin/rsyslogd -n 才是最完整的！

范例二：强制终止所有以 httpd 启动的程序 （其实并没有此程序在系统内）
[root@study ~]# killall -9 httpd

范例三：依次询问每个 bash 程序是否需要被终止运行！
[root@study ~]# killall -i -9 bash
Signal bash（13888） ？ （y/N） n &lt;==这个不杀！
Signal bash（13928） ？ （y/N） n &lt;==这个不杀！
Signal bash（13970） ？ （y/N） n &lt;==这个不杀！
Signal bash（14836） ？ （y/N） y &lt;==这个杀掉！
# 具有互动的功能！可以询问你是否要删除 bash 这个程序。要注意，若没有 -i 的参数，
# 所有的 bash 都会被这个 root 给杀掉！包括 root 自己的 bash 喔！ ^_^
```

总之，要删除某个程序，我们可以使用 PID 或者是启动该程序的指令名称， 而如果要删除某个服务呢？呵呵！最简单的方法就是利用 killall ， 因为他可以将系统当中所有以某个指令名称启动的程序全部删除。 举例来说，上面的范例二当中，系统内所有以 httpd 启动的程序，就会通通的被删除啦！ ^_^

16.3.3 关于程序的执行顺序

我们知道 Linux 是多用户多任务的环境，由 top 的输出结果我们也发现， 系统同时间有非常多的程序在运行中，只是绝大部分的程序都在休眠 （sleeping） 状态而已。 想一想，如果所有的程序同时被唤醒，那么 CPU 应该要先处理那个程序呢？也就是说，那个程序被执行的优先序比较高？ 这就得要考虑到程序的优先执行序 （Priority） 与 CPU 调度啰！



Tips CPU 调度与前一章的例行性工作调度并不一样。 CPU 调度指的是每支程序被 CPU 运行的演算规则， 而例行性工作调度则是将某支程序安排在某个时间再交由系统执行。 CPU 调度与操作系统较具有相关性！

- Priority 与 Nice 值

我们知道 CPU 一秒钟可以运行多达数 G 的微指令次数，通过核心的 CPU 调度可以让各程序被 CPU 所切换运行， 因此每个程序在一秒钟内或多或少都会被 CPU 执行部分的指令码。如果程序都是集中在一个伫列中等待 CPU 的运行， 而不具有优先顺序之分，也就是像我们去游乐场玩热门游戏需要排队一样，每个人都是照顺序来！ 你玩过一遍后还想再玩 （没有执行完毕），请到后面继续排队等待。情况有点像下面这样：

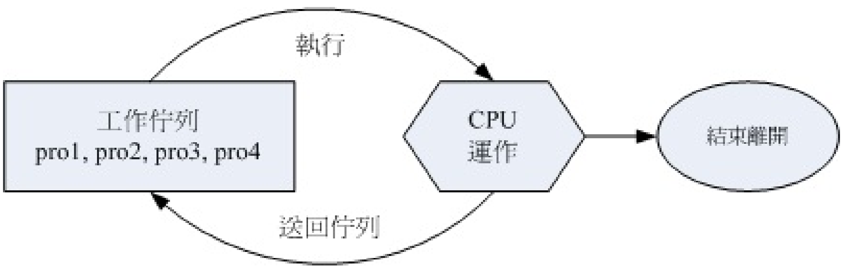


图16.3.1、并没有优先顺序的程序伫列示意图

上图中假设 pro1, pro2 是紧急的程序， pro3, pro4 是一般的程序，在这样的环境中，由于不具有优先顺序， 唉啊！pro1, pro2 还是得要继续等待而没有优待呢！如果 pro3, pro4 的工作又臭又长！那么紧急的 pro1, pro2 就得要等待个老半天才能够完成！真麻烦啊！所以啰，我们想要将程序分优先顺序啦！如果优先序较高则运行次数可以较多次， 而不需要与较慢优先的程序抢位置！我们可以将程序的优先顺序与 CPU 调度进行如下图的解释：

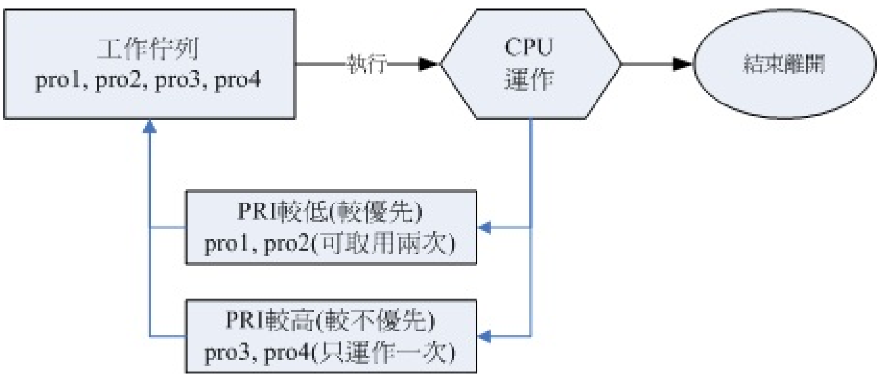


图16.3.2、具有优先顺序的程序伫列示意图

如上图所示，具高优先权的 pro1, pro2 可以被取用两次，而较不重要的 pro3, pro4 则运行次数较少。 如此一来 pro1, pro2 就可以较快被完成啦！要注意，上图仅是示意图，并非较优先者一定会被运行两次啦！ 为了要达到上述的功能，我们 Linux 给予程序一个所谓的“优先执行

序（priority, PRI）”， 这个 PRI 值越低代表越优先的意思。不过这个 PRI 值是由核心动态调整的， 使用者无法直接调整 PRI 值的。先来瞧瞧 PRI 曾在哪里出现？

```
[root@study ~]# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 14836 14835  0  90  10 - 29068 wait   pts/0    00:00:00 bash
0 R     0 19848 14836  0  90  10 - 30319 -      pts/0    00:00:00 ps
# 你应该要好奇，怎么我的 NI 已经是 10 了？还记得刚刚 top 的测试吗？我们在那边就有改过一次喔！
```

由于 PRI 是核心动态调整的，我们使用者也无权去干涉 PRI ！那如果你想要调整程序的优先执行序时，就得要通过 Nice 值了！Nice 值就是上表的 NI 啦！一般来说， PRI 与 NI 的相关性如下：

> PRI（new） = PRI（old） + nice

不过你要特别留意到，如果原本的 PRI 是 50 ，并不是我们给予一个 nice = 5 ，就会让 PRI 变成 55 喔！ 因为 PRI 是系统“动态”决定的，所以，虽然 nice 值是可以影响 PRI ，不过， 最终的 PRI 仍是要经过系统分析后才会决定的。另外， nice 值是有正负的喔，而既然 PRI 越小越早被执行， 所以，当 nice 值为负值时，那么该程序就会降低 PRI 值，亦即会变的较优先被处理。此外，你必须要留意到：

- nice 值可调整的范围为 -20 ~ 19 ；
- root 可随意调整自己或他人程序的 Nice 值，且范围为 -20 ~ 19 ；
- 一般使用者仅可调整自己程序的 Nice 值，且范围仅为 0 ~ 19 （避免一般用户抢占系统资源）；
- 一般使用者仅可将 nice 值越调越高，例如本来 nice 为 5 ，则未来仅能调整到大于 5；

这也就是说，要调整某个程序的优先执行序，就是“调整该程序的 nice 值”啦！那么如何给予某个程序 nice 值呢？有两种方式，分别是：

- 一开始执行程序就立即给予一个特定的 nice 值：用 nice 指令；
- 调整某个已经存在的 PID 的 nice 值：用 renice 指令。

- nice ：新执行的指令即给予新的 nice 值

```
[root@study ~]# nice [-n 数字] command
选项与参数：
-n  ：后面接一个数值，数值的范围 -20 ~ 19。

范例一：用 root 给一个 nice 值为 -5 ，用于执行 vim ，并观察该程序！
[root@study ~]# nice -n -5 vim &
[1] 19865
[root@study ~]# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 14836 14835  0  90  10 - 29068 wait   pts/0    00:00:00 bash
4 T     0 19865 14836  0  85   5 - 37757 signal pts/0    00:00:00 vim
0 R     0 19866 14836  0  90  10 - 30319 -      pts/0    00:00:00 ps
# 原本的 bash PRI 为 90  ，所以 vim 默认应为 90。不过由于给予 nice  为 -5 ，
# 因此 vim 的 PRI 降低了！RPI 与 NI 各减 5 ！但不一定每次都是正好相同喔！因为核心会动态调整

[root@study ~]# kill -9 %1 &lt;==测试完毕将 vim 关闭
```

就如同前面说的， nice 是用来调整程序的执行优先顺序！这里只是一个执行的范例罢了！ 通常什么时候要将 nice 值调大呢？举例来说，系统的背景工作中， 某些比较不重要的程序之进行：例如备份工作！由于备份工作相当的耗系统资源， 这个时候就可以将备份的指令之 nice 值调大一些，可以使系统的资源分配的更为公平！

- renice ：已存在程序的 nice 重新调整

```
[root@study ~]# renice [number] PID
选项与参数：
PID ：某个程序的 ID 啊！

范例一：找出自己的 bash PID ，并将该 PID 的 nice 调整到 -5
[root@study ~]# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 14836 14835  0  90  10 - 29068 wait   pts/0    00:00:00 bash
0 R     0 19900 14836  0  90  10 - 30319 -      pts/0    00:00:00 ps

[root@study ~]# renice -5 14836
14836 （process ID） old priority 10, new priority -5

[root@study ~]# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 14836 14835  0  75  -5 - 29068 wait   pts/0    00:00:00 bash
0 R     0 19910 14836  0  75  -5 - 30319 -      pts/0    00:00:00 ps
```

如果要调整的是已经存在的某个程序的话，那么就得要使用 renice 了。使用的方法很简单，renice 后面接上数值及 PID 即可。因为后面接的是 PID ，所以你务必要以 ps 或者其他程序观察的指令去找出 PID 才行啊！

由上面这个范例当中我们也看的出来，虽然修改的是 bash 那个程序，但是该程序所触发的 ps 指令当中的 nice 也会继承而为 -5 喔！了解了吧！整个 nice 值是可以在父程序 --> 子程序之间传递的呢！ 另外，除了 renice 之外，其实那个 top 同样的也是可以调整 nice 值的！

16.3.4 系统资源的观察

除了系统的程序之外，我们还必须就系统的一些资源进行检查啊！举例来说，我们使用 top 可以看到很多系统的资源对吧！那么，还有没有其他的工具可以查阅的？ 当然有啊！下面这些工具指令可以玩一玩！

- free ：观察内存使用情况

```
[root@study ~]# free [-b&#124;-k&#124;-m&#124;-g&#124;-h] [-t] [-s N -c N]
选项与参数：
-b  ：直接输入 free 时，显示的单位是 KBytes，我们可以使用 b（Bytes）, m（MBytes）
      k（KBytes）, 及 g（GBytes） 来显示单位喔！也可以直接让系统自己指定单位 （-h）
-t  ：在输出的最终结果，显示实体内存与 swap 的总量。
-s  ：可以让系统每几秒钟输出一次，不间断的一直输出的意思！对于系统观察挺有效！
-c  ：与 -s 同时处理～让 free 列出几次的意思～

范例一：显示目前系统的内存容量
[root@study ~]# free -m
              total        used        free      shared  buff/cache   available
Mem:           2848         346        1794           8         706        2263
Swap:          1023           0        1023
```

仔细看看，我的系统当中有 2848MB 左右的实体内存，我的 swap 有 1GB 左右， 那我使用 free -m 以 MBytes 来显示时，就会出现上面的信息。Mem 那一行显示的是实体内存的量，Swap 则是内存交换空间的量。 total 是总量， used 是已被使用的量， free 则是剩余可用的量。 后面的 shared/buffers/cached 则是在已被使用的量当中，用来作为缓冲及高速缓存的量，这些 shared/buffers/cached 的用量中，在系统比较忙碌时， 可以被释出而继续利用！因此后面就有一个 available （可用的） 数值！ 。

请看上头范例一的输出，我们可以发现这部测试机根本没有什么特别的服务，但是竟然有 706MB 左右的 cache 耶！ 因为鸟哥在测试过程中还是有读/写/执行很多的文件嘛！这些文件就会被系统暂时高速缓存下来，等待下次运行时可以更快速的取出之意！ 也就是说，系统是“很有效率的将所有的内存用光光”，目的是为了让系统的存取性能加速啦！

很多朋友都会问到这个问题“我的系统明明很轻松，为何内存会被用光光？”现在瞭了吧？ 被用光是正常的！而需要注意的反而是 swap 的量。一般来说， swap 最好不要被使用，尤其 swap 最好不要被使用超过 20% 以上， 如果您发现 swap 的用量超过 20% ，那么，最好还是买实体内存来插吧！ 因为， Swap 的性能跟实体内存实在差很多，而系统会使用到 swap ，绝对是因为实体内存不足了才会这样做的！如此，了解吧！



Tips Linux 系统为了要加速系统性能，所以会将最常使用到的或者是最近使用到的文件数据高速缓存 （cache） 下来， 这样未来系统要使用该文件时，就直接由内存中搜寻取出，而不需要重新读取硬盘，速度上面当然就加快了！ 因此，实体内存被用光是正常的喔！

- uname：查阅系统与核心相关信息

```
[root@study ~]# uname [-asrmpi]
选项与参数：
-a  ：所有系统相关的信息，包括下面的数据都会被列出来；
-s  ：系统核心名称
-r  ：核心的版本
-m  ：本系统的硬件名称，例如 i686 或 x86_64 等；
-p  ：CPU 的类型，与 -m 类似，只是显示的是 CPU 的类型！
-i  ：硬件的平台 （ix86）

范例一：输出系统的基本信息
[root@study ~]# uname -a
Linux study.centos.vbird 3.10.0-229.el7.x86_64 #1 SMP Fri Mar 6 11:36:42 UTC 2015
x86_64 x86_64 x86_64 GNU/Linux
```

这个咚咚我们前面使用过很多次了喔！uname 可以列出目前系统的核心版本、 主要硬件平台以及 CPU 类型等等的信息。以上面范例一的状态来说，我的 Linux 主机使用的核心名称为 Linux，而主机名称为 study.centos.vbird，核心的版本为 3.10.0-229.el7.x86_64 ，该核心版本创建的日期为 2015-3-6，适用的硬件平台为 x86_64 以上等级的硬件平台喔。

- uptime：观察系统启动时间与工作负载

这个指令很单纯呢！就是显示出目前系统已经开机多久的时间，以及 1, 5, 15 分钟的平均负载就是了。还记得 top 吧？没错啦！这个 uptime 可以显示出 top 画面的最上面一行！

```
[root@study ~]# uptime
 02:35:27 up  7:48,  3 users,  load average: 0.00, 0.01, 0.05
# [top](../Text/index.html#topm) 这个指令已经谈过相关信息，不再聊！
```

- netstat ：追踪网络或插槽档

这个 netstat 也是挺好玩的，其实这个指令比较常被用在网络的监控方面，不过，在程序管理方面也是需要了解的啦！ 这个指令的执行如下所示：基本上， netstat 的输出分为两大部分，分别是网络与系统自己的程序相关性部分：

```
[root@study ~]# netstat -[atunlp]
选项与参数：
-a  ：将目前系统上所有的连线、监听、Socket 数据都列出来
-t  ：列出 tcp 网络封包的数据
-u  ：列出 udp 网络封包的数据
-n  ：不以程序的服务名称，以埠号 (port number) 来显示；
-l  ：列出目前正在网络监听 (listen) 的服务；
-p  ：列出该网络服务的程序 PID

范例一：列出目前系统已经创建的网络连线与 unix socket 状态
[root@study ~]# netstat
Active Internet connections (w/o servers) &lt;==与网络较相关的部分
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 172.16.15.100:ssh       172.16.220.234:48300    ESTABLISHED
Active UNIX domain sockets (w/o servers)  &lt;==与本机的程序自己的相关性(非网络)
Proto RefCnt Flags       Type       State         I-Node   Path
unix  2      [ ]         DGRAM                    1902     @/org/freedesktop/systemd1/not
unix  2      [ ]         DGRAM                    1944     /run/systemd/shutdownd
....(中间省略)....
unix  3      [ ]         STREAM     CONNECTED     25425    @/tmp/.X11-unix/X0
unix  3      [ ]         STREAM     CONNECTED     28893
unix  3      [ ]         STREAM     CONNECTED     21262
```

在上面的结果当中，显示了两个部分，分别是网络的连线以及 linux 上面的 socket 程序相关性部分。 我们先来看看网际网络连线情况的部分：

- Proto ：网络的封包协定，主要分为 TCP 与 UDP 封包，相关数据请参考服务器篇 ；
- Recv-Q：非由使用者程序链接到此 socket 的复制的总 Bytes 数；
- Send-Q：非由远端主机传送过来的 acknowledged 总 Bytes 数；
- Local Address ：本地端的 IP:port 情况
- Foreign Address：远端主机的 IP:port 情况
- State ：连线状态，主要有创建（ESTABLISED）及监听（LISTEN）；

我们看上面仅有一条连线的数据，他的意义是："通过 TCP 封包的连线，远端的 172.16.220.234:48300 连线到本地端的 172.16.15.100:ssh ，这条连线状态是创建（ESTABLISHED） 的状态！"至于更多的网络环境说明， 就得到鸟哥的另一本服务器篇查阅啰！

除了网络上的连线之外，其实 Linux 系统上面的程序是可以接收不同程序所发送来的信息，那就是 Linux 上头的插槽档 （socket file）。我们在第五章的文件种类有稍微提到 socket 文件， 但当时未谈到程序的概念，所以没有深入谈论。socket file 可以沟通两个程序之间的信息，因此程序可以取得对方传送过来的数据。 由于有 socket file，因此类似 X Window 这种需要通过网络连接的软件，目前新版的 distributions 就以 socket 来进行窗口接口的连线沟通了。上表中 socket file 的输出字段有：

- Proto ：一般就是 unix 啦；
- RefCnt：连接到此 socket 的程序数量；
- Flags ：连线的旗标；
- Type ：socket 存取的类型。主要有确认连线的 STREAM 与不需确认的 DGRAM 两种；
- State ：若为 CONNECTED 表示多个程序之间已经连线创建。
- Path ：连接到此 socket 的相关程序的路径！或者是相关数据输出的路径。

以上表的输出为例，最后那三行在 /tmp/.xx 下面的数据，就是 X Window 窗口接口的相关程序啦！ 而 PATH 指向的就是这些程序要交换数据的插槽文件啰！好！那么 netstat 可以帮我们进行什么任务呢？ 很多喔！我们先来看看，利用 netstat 去看看我们的哪些程序有启动哪些网络的“后门”呢？

```
范例二：找出目前系统上已在监听的网络连线及其 PID
[root@study ~]# netstat -tulnp
Active Internet connections （only servers）
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Progr
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      1326/sshd
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      2349/mast
tcp6       0      0 :::22                   :::*                    LISTEN      1326/sshd
tcp6       0      0 ::1:25                  :::*                    LISTEN      2349/mast
udp        0      0 0.0.0.0:123             0.0.0.0:*                           751/chron
udp        0      0 127.0.0.1:323           0.0.0.0:*                           751/chron
udp        0      0 0.0.0.0:57808           0.0.0.0:*                           743/avahi
udp        0      0 0.0.0.0:5353            0.0.0.0:*                           743/avahi
udp6       0      0 :::123                  :::*                                751/chron
udp6       0      0 ::1:323                 :::*                                751/chron
# 除了可以列出监听网络的接口与状态之外，最后一个字段还能够显示此服务的
# PID 号码以及程序的指令名称喔！例如上头的 1326 就是该 PID

范例三：将上述的 0.0.0.0:57808 那个网络服务关闭的话？
[root@study ~]# kill -9 743
[root@study ~]# killall -9 avahi-daemon
```

很多朋友常常有疑问，那就是，我的主机目前到底开了几个门（ports）！其实，不论主机提供什么样的服务， 一定必须要有相对应的 program 在主机上面执行才行啊！举例来说，我们鸟园的 Linux 主机提供的就是 WWW 服务，那么我的主机当然有一个程序在提供 WWW 的服务啊！那就是 Apache 这个软件所提供的啦！ ^_^。 所以，当我执行了这个程序之后，我的系统自然就可以提供 WWW 的服务了。那如何关闭啊？ 就关掉该程序所触发的那个程序就好了！例如上面的范例三所提供的例子啊！不过，这个是非正规的作法喔！ 正规的作法，请查阅下一章的说明呦！

- dmesg ：分析核心产生的讯息

系统在开机的时候，核心会去侦测系统的硬件，你的某些硬件到底有没有被捉到，那就与这个时候的侦测有关。 但是这些侦测的过程要不是没有显示在屏幕上，就是很飞快的在屏幕上一闪而逝！能不能把核心侦测的讯息捉出来瞧瞧？ 可以的，那就使用 dmesg 吧！

所有核心侦测的讯息，不管是开机时候还是系统运行过程中，反正只要是核心产生的讯息，都会被记录到内存中的某个保护区段。 dmesg 这个指令就能够将该区段的讯息读出来的！因为讯息实在太多了，所以执行时可以加入这个管线指令“ | more ”来使画面暂停！

```
范例一：输出所有的核心开机时的信息
[root@study ~]# dmesg &#124; more

范例二：搜寻开机的时候，硬盘的相关信息为何？
[root@study ~]# dmesg &#124; grep -i vda
[    0.758551]  vda: vda1 vda2 vda3 vda4 vda5 vda6 vda7 vda8 vda9
[    3.964134] XFS （vda2）: Mounting V4 Filesystem
....（下面省略）....
```

由范例二就知道我这部主机的硬盘的格式是什么了吧！

- vmstat ：侦测系统资源变化

如果你想要动态的了解一下系统资源的运行，那么这个 vmstat 确实可以玩一玩！vmstat 可以侦测“ CPU / 内存 / 磁盘输入输出状态 ”等等，如果你想要了解一部繁忙的系统到底是哪个环节最累人， 可以使用 vmstat 分析看看。下面是常见的选项与参数说明：

```
[root@study ~]# vmstat [-a] [延迟 [总计侦测次数]] &lt;==CPU/内存等信息
[root@study ~]# vmstat [-fs]                      &lt;==内存相关
[root@study ~]# vmstat [-S 单位]                  &lt;==设置显示数据的单位
[root@study ~]# vmstat [-d]                       &lt;==与磁盘有关
[root@study ~]# vmstat [-p 分区]                  &lt;==与磁盘有关
选项与参数：
-a   ：使用 inactive/active（活跃与否） 取代 buffer/cache 的内存输出信息；
-f   ：开机到目前为止，系统复制 （fork） 的程序数；
-s   ：将一些事件 （开机至目前为止） 导致的内存变化情况列表说明；
-S   ：后面可以接单位，让显示的数据有单位。例如 K/M 取代 Bytes 的容量；
-d   ：列出磁盘的读写总量统计表
-p   ：后面列出分区，可显示该分区的读写总量统计表

范例一：统计目前主机 CPU 状态，每秒一次，共计三次！
[root@study ~]# vmstat 1 3
procs ------------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 1838092   1504 722216    0    0     4     1    6    9  0  0 100  0  0
 0  0      0 1838092   1504 722200    0    0     0     0   13   23  0  0 100  0  0
 0  0      0 1838092   1504 722200    0    0     0     0   25   46  0  0 100  0  0
```

利用 vmstat 甚至可以进行追踪喔！你可以使用类似“ vmstat 5 ”代表每五秒钟更新一次，且无穷的更新！直到你按下 [ctrl]-c 为止。如果你想要实时的知道系统资源的运行状态，这个指令就不能不知道！那么上面的表格各项字段的意义为何？ 基本说明如下：

- 程序字段 （procs） 的项目分别为： r ：等待运行中的程序数量；b：不可被唤醒的程序数量。这两个项目越多，代表系统越忙碌 （因为系统太忙，所以很多程序就无法被执行或一直在等待而无法被唤醒之故）。

- 内存字段（memory） 项目分别为： swpd：虚拟内存被使用的容量； free：未被使用的内存容量； buff：用于缓冲内存； cache：用于高速缓存内存。 这部份则与 free 是相同的。
- 内存交换空间（swap） 的项目分别为： si：由磁盘中将程序取出的量； so：由于内存不足而将没用到的程序写入到磁盘的 swap 的容量。 如果 si/so 的数值太大，表示内存内的数据常常得在磁盘与内存之间传来传去，系统性能会很差！
- 磁盘读写（io） 的项目分别为： bi：由磁盘读入的区块数量； bo：写入到磁盘去的区块数量。如果这部份的值越高，代表系统的 I/O 非常忙碌！
- 系统（system） 的项目分别为： in：每秒被中断的程序次数； cs：每秒钟进行的事件切换次数；这两个数值越大，代表系统与周边设备的沟通非常频繁！ 这些周边设备当然包括磁盘、网卡、时间钟等。
- CPU 的项目分别为： us：非核心层的 CPU 使用状态； sy：核心层所使用的 CPU 状态； id：闲置的状态； wa：等待 I/O 所耗费的 CPU 状态； st：被虚拟机（virtual machine） 所盗用的 CPU 使用状态（2.6.11 以后才支持）。

由于鸟哥的机器是测试机，所以并没有什么 I/O 或者是 CPU 忙碌的情况。如果改天你的服务器非常忙碌时， 记得使用 vmstat 去看看，到底是哪个部分的资源被使用的最为频繁！一般来说，如果 I/O 部分很忙碌的话，你的系统会变的非常慢！ 让我们再来看看，那么磁盘的部分该如何观察：

```
范例二：系统上面所有的磁盘的读写状态
[root@study ~]# vmstat -d
disk- ------------reads------------ ------------writes----------- -----IO------
       total merged sectors      ms  total merged sectors      ms    cur    sec
vda    21928      0  992587   47490   7239   2225  258449   13331      0     26
sda      395      1    3168     213      0      0       0       0      0      0
sr0        0      0       0       0      0      0       0       0      0      0
dm-0   19139      0  949575   44608   7672      0  202251   16264      0     25
dm-1     336      0    2688     327      0      0       0       0      0      0
md0      212      0    1221       0     14      0    4306       0      0      0
dm-2     218      0    9922     565     54      0    4672     128      0      0
dm-3     179      0     957     182     11      0    4306      68      0      0
```

详细的各字段就请诸位大德查阅一下 man vmstat 啰！反正与读写有关啦！这样了解乎！

16.4 特殊文件与程序

我们在第六章曾经谈到特殊权限的 SUID/SGID/SBIT ，虽然第六章已经将这三种特殊权限作了详细的解释，不过，我们依旧要来探讨的是，那么到底这些权限对于你的“程序”是如何影响的？ 此外，程序可能会使用到系统资源，举例来说，磁盘就是其中一项资源。哪天你在 umount 磁盘时，系统老是出现“ device is busy ”的字样～到底是怎么回事啊？我们下面就来谈一谈这些和程序有关系的细节部分：

16.4.1 具有 SUID/SGID 权限的指令执行状态

SUID 的权限其实与程序的相关性非常的大！为什么呢？先来看看 SUID 的程序是如何被一般使用者执行，且具有什么特色呢？

- SUID 权限仅对二进制程序（binary program）有效；
- 执行者对于该程序需要具有 x 的可执行权限；
- 本权限仅在执行该程序的过程中有效 （run-time）；
- 执行者将具有该程序拥有者 （owner） 的权限。

所以说，整个 SUID 的权限会生效是由于“具有该权限的程序被触发”，而我们知道一个程序被触发会变成程序， 所以啰，执行者可以具有程序拥有者的权限就是在该程序变成程序的那个时候啦！第六章我们还没谈到程序的概念， 所以你或许那时候会觉得很奇怪，为啥执行了 passwd 后你就具有 root 的权限呢？不都是一般使用者执行的吗？ 这是因为你在触发 passwd 后，会取得一个新的程序与 PID，该 PID 产生时通过 SUID 来给予该 PID 特殊的权限设置啦！ 我们使用 dmtsai 登陆系统且执行 passwd 后，通过工作控制来理解一下！

```
[dmtsai@study ~]$ passwd
Changing password for user dmtsai.
Changing password for dmtsai
（current） UNIX password: &lt;==这里按下 [ctrl]-z 并且按下 [enter]
[1]+  Stopped                 passwd

[dmtsai@study ~]$ pstree -uA
systemd-+-ModemManager---2*[{ModemManager}]
....（中间省略）....
        &#124;-sshd---sshd---sshd（dmtsai）---bash-+-passwd（root）
        &#124;                                      `-pstree
....（下面省略）....
```

从上表的结果我们可以发现，底线的部分是属于 dmtsai 这个一般帐号的权限，特殊字体的则是 root 的权限！ 但你看到了， passwd 确实是由 bash 衍生出来的！不过就是权限不一样！ 通过这样的解析， 你也会比较清楚为何不同程序所产生的权限不同了吧！这是由于“SUID 程序运行过程中产生的程序”的关系啦！

那么既然 SUID/SGID 的权限是比较可怕的，您该如何查询整个系统的 SUID/SGID 的文件呢？ 应该是还不会忘记吧？使用 find 即可啊！

> find / -perm /6000

16.4.2 /proc/* 代表的意义

其实，我们之前提到的所谓的程序都是在内存当中嘛！而内存当中的数据又都是写入到 /proc/* 这个目录下的，所以啰，我们当然可以直接观察 /proc 这个目录当中的文件啊！ 如果你观察过 /proc 这个目录的话，应该会发现他有点像这样：

```
[root@study ~]# ll /proc
dr-xr-xr-x.  8 root          root              0 Aug  4 18:46 1
dr-xr-xr-x.  8 root          root              0 Aug  4 18:46 10
dr-xr-xr-x.  8 root          root              0 Aug  4 18:47 10548
....（中间省略）....
-r--r--r--.  1 root          root              0 Aug  5 17:48 uptime
-r--r--r--.  1 root          root              0 Aug  5 17:48 version
-r--------.  1 root          root              0 Aug  5 17:48 vmallocinfo
-r--r--r--.  1 root          root              0 Aug  5 17:48 vmstat
-r--r--r--.  1 root          root              0 Aug  5 17:48 zoneinfo
```

基本上，目前主机上面的各个程序的 PID 都是以目录的型态存在于 /proc 当中。 举例来说，我们开机所执行的第一支程序 systemd 他的 PID 是 1 ， 这个 PID 的所有相关信息都写入在 /proc/1/* 当中！若我们直接观察 PID 为 1 的数据好了，他有点像这样：

```
[root@study ~]# ll /proc/1
dr-xr-xr-x. 2 root root 0 Aug  4 19:25 attr
-rw-r--r--. 1 root root 0 Aug  4 19:25 autogroup
-r--------. 1 root root 0 Aug  4 19:25 auxv
-r--r--r--. 1 root root 0 Aug  4 18:46 cgroup
--w-------. 1 root root 0 Aug  4 19:25 clear_refs
-r--r--r--. 1 root root 0 Aug  4 18:46 cmdline  &lt;==就是指令串
-r--------. 1 root root 0 Aug  4 18:46 environ  &lt;==一些环境变量
lrwxrwxrwx. 1 root root 0 Aug  4 18:46 exe
....（以下省略）....
```

里面的数据还挺多的，不过，比较有趣的其实是两个文件，分别是：

- cmdline：这个程序被启动的指令串；
- environ：这个程序的环境变量内容。

很有趣吧！如果你查阅一下 cmdline 的话，就会发现：

```
[root@study ~]# cat /proc/1/cmdline
/usr/lib/systemd/systemd--switched-root--system--deserialize24
```

就是这个指令、选项与参数启动 systemd 的啦！这还是跟某个特定的 PID 有关的内容呢，如果是针对整个 Linux 系统相关的参数呢？那就是在 /proc 目录下面的文件啦！相关的文件与对应的内容是这样的： [3]

文件名	文件内容
/proc/cmdline	载入 kernel 时所下达的相关指令与参数！查阅此文件，可了解指令是如何启动的！
/proc/cpuinfo	本机的 CPU 的相关信息，包含频率、类型与运算功能等
/proc/devices	这个文件记录了系统各个主要设备的主要设备代号，与 mknod 有关呢！
/proc/filesystems	目前系统已经载入的文件系统啰！
/proc/interrupts	目前系统上面的 IRQ 分配状态。
/proc/ioports	目前系统上面各个设备所配置的 I/O 位址。
/proc/kcore	这个就是内存的大小啦！好大对吧！但是不要读他啦！
/proc/loadavg	还记得 top 以及 uptime 吧？没错！上头的三个平均数值就是记录在此！
/proc/meminfo	使用 free 列出的内存信息，嘿嘿！在这里也能够查阅到！
/proc/modules	目前我们的 Linux 已经载入的模块列表，也可以想成是驱动程序啦！
/proc/mounts	系统已经挂载的数据，就是用 mount 这个指令调用出来的数据啦！
/proc/swaps	到底系统挂载入的内存在哪里？呵呵！使用掉的 partition 就记录在此啦！
/proc/partitions	使用 fdisk -l 会出现目前所有的 partition 吧？在这个文件当中也有纪录喔！
/proc/uptime	就是用 uptime 的时候，会出现的信息啦！
/proc/version	核心的版本，就是用 uname -a 显示的内容啦！
/proc/bus/*	一些总线的设备，还有 USB 的设备也记录在此喔！

其实，上面这些文件鸟哥在此建议您可以使用 cat 去查阅看看，不必深入了解， 不过，观看过文件内容后，毕竟会比较有感觉啦！如果未来您想要自行撰写某些工具软件， 那么这个目录下面的相关文件可能会对您有点帮助的喔！

16.4.3. 查询已打开文件或已执行程序打开之文件

其实还有一些与程序相关的指令可以值得参考与应用的，我们来谈一谈：

- fuser：借由文件（或文件系统）找出正在使用该文件的程序

有的时候我想要知道我的程序到底在这次启动过程中打开了多少文件，可以利用 fuser 来观察啦！ 举例来说，你如果卸载时发现系统通知：“ device is busy ”，那表示这个文件系统正在忙碌中， 表示有某支程序有利用到该文件系统啦！那么你就可以利用 fuser 来追踪啰！fuser 语法有点像这样：

```
[root@study ~]# fuser [-umv] [-k [i] [-signal]] file/dir
选项与参数：
-u   ：除了程序的 PID 之外，同时列出该程序的拥有者；
-m   ：后面接的那个文件名会主动的上提到该文件系统的最顶层，对 umount 不成功很有效！
-v   ：可以列出每个文件与程序还有指令的完整相关性！
-k   ：找出使用该文件/目录的 PID ，并试图以 SIGKILL 这个讯号给予该 PID；
-i   ：必须与 -k 配合，在删除 PID 之前会先询问使用者意愿！
-signal：例如 -1 -15 等等，若不加的话，默认是 SIGKILL （-9） 啰！

范例一：找出目前所在目录的使用 PID/所属帐号/权限 为何？
[root@study ~]# fuser -uv .
                     USER        PID ACCESS COMMAND
/root:               root      13888 ..c..  （root）bash
                     root      31743 ..c..  （root）bash
```

看到输出的结果没？他说“.”下面有两个 PID 分别为 13888, 31743 的程序，该程序属于 root 且指令为 bash 。 比较有趣的是那个 ACCESS 的项目，那个项目代表的意义为：

- c ：此程序在当前的目录下（非次目录）；
- e ：可被触发为执行状态；
- f ：是一个被打开的文件；
- r ：代表顶层目录 （root directory）；
- F ：该文件被打开了，不过在等待回应中；
- m ：可能为分享的动态函数库；

那如果你想要查阅某个文件系统下面有多少程序正在占用该文件系统时，那个 -m 的选项就很有帮助了！ 让我们来做几个简单的测试，包括实体的文件系统挂载与 /proc 这个虚拟文件系统的内容， 看看有多少的程序对这些挂载点或其他目录的使用状态吧！

```
范例二：找到所有使用到 /proc 这个文件系统的程序吧！
[root@study ~]# fuser -uv /proc
/proc:               root     kernel mount （root）/proc
                     rtkit       768 .rc.. （rtkit）rtkit-daemon
# 数据量还不会很多，虽然这个目录很繁忙～没关系！我们可以继续这样作，看看其他的程序！

[root@study ~]# fuser -mvu /proc
                     USER        PID ACCESS COMMAND
/proc:               root     kernel mount （root）/proc
                     root          1 f.... （root）systemd
                     root          2 ...e. （root）kthreadd
.....（下面省略）.....
# 有这几支程序在进行 /proc 文件系统的存取喔！这样清楚了吗？

范例三：找到所有使用到 /home 这个文件系统的程序吧！
[root@study ~]# echo $$
31743  # 先确认一下，自己的 bash PID 号码吧！
[root@study ~]# cd /home
[root@study home]# fuser -muv .
                     USER        PID ACCESS COMMAND
/home:               root     kernel mount （root）/home
                     dmtsai    31535 ..c.. （dmtsai）bash
                     root      31571 ..c.. （root）passwd
                     root      31737 ..c.. （root）sudo
                     root      31743 ..c.. （root）bash    # 果然，自己的 PID 在啊！
[root@study home]# cd ~
[root@study ~]# umount /home
umount: /home: target is busy.
        （In some cases useful info about processes that use
         the device is found by lsof（8） or fuser（1））
# 从 fuser 的结果可以知道，总共有五只 process 在该目录下运行，那即使 root 离开了 /home，
# 当然还是无法 umount 的！那要怎办？哈哈！可以通过如下方法一个一个删除～
[root@study ~]# fuser -mki /home
/home:               31535c 31571c 31737c  # 你会发现， PID 跟上面查到的相同！
Kill process 31535 ? （y/N） # 这里会问你要不要删除！当然不要乱删除啦！通通取消！
```

既然可以针对整个文件系统，那么能不能仅针对单一文件啊？当然可以啰！看一下下面的案例先：

```
范例四：找到 /run 下面属于 FIFO 类型的文件，并且找出存取该文件的程序
[root@study ~]# find /run -type p
.....（前面省略）.....
/run/systemd/sessions/165.ref
/run/systemd/sessions/1.ref
/run/systemd/sessions/c1.ref   # 随便抓个项目！就是这个好了！来测试一下！

[root@study ~]# fuser -uv /run/systemd/sessions/c1.ref
                     USER        PID ACCESS COMMAND
/run/systemd/sessions/c1.ref:
                     root        763 f.... （root）systemd-logind
                     root       5450 F.... （root）gdm-session-wor
# 通常系统的 FIFO 文件都会放置到 /run 下面，通过这个方式来追踪该文件被存取的 process！
# 也能够晓得系统有多忙碌啊！呵呵！
```

如何？很有趣的一个指令吧！通过这个 fuser 我们可以找出使用该文件、目录的程序，借以观察的啦！ 他的重点与 ps, pstree 不同。 fuser 可以让我们了解到某个文件 （或文件系统） 目前正在被哪些程序所利用！

- lsof ：列出被程序所打开的文件文件名

相对于 fuser 是由文件或者设备去找出使用该文件或设备的程序，反过来说，如何查出某个程序打开或者使用的文件与设备呢？呼呼！那就是使用 lsof 啰～

```
[root@study ~]# lsof [-aUu] [+d]
选项与参数：
-a  ：多项数据需要“同时成立”才显示出结果时！
-U  ：仅列出 Unix like 系统的 socket 文件类型；
-u  ：后面接 username，列出该使用者相关程序所打开的文件；
+d  ：后面接目录，亦即找出某个目录下面已经被打开的文件！

范例一：列出目前系统上面所有已经被打开的文件与设备：
[root@study ~]# lsof
COMMAND    PID   TID    USER   FD    TYPE     DEVICE  SIZE/OFF       NODE NAME
systemd      1          root  cwd     DIR      253,0      4096        128 /
systemd      1          root  rtd     DIR      253,0      4096        128 /
systemd      1          root  txt     REG      253,0   1230920     967763 /usr/lib/systemd/s
....（下面省略）....
# 注意到了吗？是的，在默认的情况下， lsof 会将目前系统上面已经打开的
# 文件全部列出来～所以，画面多的吓人啊！您可以注意到，第一个文件 systemd 执行的
# 地方就在根目录，而根目录，嘿嘿！所在的 inode 也有显示出来喔！

范例二：仅列出关于 root 的所有程序打开的 socket 文件
[root@study ~]# lsof -u root -a -U
COMMAND     PID USER   FD   TYPE             DEVICE SIZE/OFF   NODE NAME
systemd       1 root    3u  unix 0xffff8800b7756580      0t0  13715 socket
systemd       1 root    7u  unix 0xffff8800b7755a40      0t0   1902 @/org/freedesktop/sys
systemd       1 root    9u  unix 0xffff8800b7756d00      0t0   1903 /run/systemd/private
.....（中间省略）.....
Xorg       4496 root    1u  unix 0xffff8800ab107480      0t0  25981 @/tmp/.X11-unix/X0
Xorg       4496 root    3u  unix 0xffff8800ab107840      0t0  25982 /tmp/.X11-unix/X0
Xorg       4496 root   16u  unix 0xffff8800b7754f00      0t0  25174 @/tmp/.X11-unix/X0
.....（下面省略）.....
# 注意到那个 -a 吧！如果你分别输入 lsof -u root 及 lsof -U ，会有啥信息？
# 使用 lsof -u root -U 及 lsof -u root -a -U ，呵呵！都不同啦！
# -a 的用途就是在解决同时需要两个项目都成立时啊！ ^_^

范例三：请列出目前系统上面所有的被启动的周边设备
[root@study ~]# lsof +d /dev
COMMAND     PID           USER   FD   TYPE             DEVICE SIZE/OFF NODE NAME
systemd       1           root   0u   CHR                 1,3      0t0 1028 /dev/null
systemd       1           root   1u   CHR                 1,3      0t0 1028 /dev/null
# 看吧！因为设备都在 /dev 里面嘛！所以啰，使用搜寻目录即可啊！

范例四：秀出属于 root 的 bash 这支程序所打开的文件
[root@study ~]# lsof -u root &#124; grep bash
ksmtuned    781 root  txt    REG    253,0    960384   33867220 /usr/bin/bash
bash      13888 root  cwd    DIR    253,0      4096   50331777 /root
bash      13888 root  rtd    DIR    253,0      4096        128 /
bash      13888 root  txt    REG    253,0    960384   33867220 /usr/bin/bash
bash      13888 root  mem    REG    253,0 106065056   17331169 /usr/lib/locale/locale-arc
....（下面省略）....
```

这个指令可以找出您想要知道的某个程序是否有启用哪些信息？例如上头提到的范例四的执行结果呢！ ^_^

- pidof ：找出某支正在执行的程序的 PID

```
[root@study ~]# pidof [-sx] program_name
选项与参数：
-s  ：仅列出一个 PID 而不列出所有的 PID
-x  ：同时列出该 program name 可能的 PPID 那个程序的 PID

范例一：列出目前系统上面 systemd 以及 rsyslogd 这两个程序的 PID
[root@study ~]# pidof systemd rsyslogd
1 742
# 理论上，应该会有两个 PID 才对。上面的显示也是出现了两个 PID 喔。
# 分别是 systemd 及 rsyslogd 这两支程序的 PID 啦。
```

很简单的用法吧，通过这个 pidof 指令，并且配合 ps aux 与正则表达式，就可以很轻易的找到您所想要的程序内容了呢。 如果要找的是 bash ，那就 pidof bash ，立刻列出一堆 PID 号码了～

16.5 SELinux 初探

从进入了 CentOS 5.x 之后的 CentOS 版本中 （当然包括 CentOS 7），SELinux 已经是个非常完备的核心模块了！尤其 CentOS 提供了很多管理 SELinux 的指令与机制， 因此在整体架构上面是单纯且容易操作管理的！所以，在没有自行开发网络服务软件以及使用其他第三方协力软件的情况下， 也就是全部使用 CentOS 官方提供的软件来使用我们服务器的情况下，建议大家不要关闭 SELinux 了喔！ 让我们来仔细的玩玩这家伙吧！

16.5.1 什么是 SELinux

什么是 SELinux 呢？其实他是“ Security Enhanced Linux ”的缩写，字面上的意义就是安全强化的 Linux 之意！那么所谓的“安全强化”是强化哪个部分？ 是网络资安还是权限管理？下面就让我们来谈谈吧！

- 当初设计的目标：避免资源的误用

SELinux 是由美国国家安全局 （NSA） 开发的，当初开发这玩意儿的目的是因为很多企业界发现， 通常系统出现问题的原因大部分都在于“内部员工的资源误用”所导致的，实际由外部发动的攻击反而没有这么严重。 那么什么是“员工资源误用”呢？举例来说，如果有个不是很懂系统的系统管理员为了自己设置的方便，将网页所在目录 /var/www/html/ 的权限设置为 drwxrwxrwx 时，你觉得会有什么事情发生？

现在我们知道所有的系统资源都是通过程序来进行存取的，那么 /var/www/html/ 如果设置为 777 ， 代表所有程序均可对该目录存取，万一你真的有启动 WWW 服务器软件，那么该软件所触发的程序将可以写入该目录， 而该程序却是对整个 Internet 提供服务的！只要有心人接触到这支程序，而且该程序刚好又有提供使用者进行写入的功能， 那么外部的人很可能就会对你的系统写入些莫名其妙的东西！那可真是不得了！一个小小的 777 问题可是大大的！

为了控管这方面的权限与程序的问题，所以美国国家安全局就着手处理操作系统这方面的控管。 由于 Linux 是自由软件，程序码都是公开的，因此她们便使用 Linux 来作为研究的目标， 最后更将研究的结果整合到 Linux 核心里面去，那就是 SELinux 啦！所以说， SELinux 是整合到核心的一个模块喔！ 更多的 SELinux 相关说明可以参考：

- http://www.nsa.gov/research/selinux/

这也就是说：其实 SELinux 是在进行程序、文件等细部权限设置依据的一个核心模块！ 由于启动网络服务的也是程序，因此刚好也能够控制网络服务能否存取系统资源的一道关卡！ 所以，在讲到 SELinux 对系统的存取控制之前，我们得先来回顾一下之前谈到的系统文件权限与使用者之间的关系。 因为先谈完这个你才会知道为何需要 SELinux 的啦！

- 传统的文件权限与帐号关系：自主式存取控制, DAC

我们第十三章的内容，知道系统的帐号主要分为系统管理员 （root） 与一般用户，而这两种身份能否使用系统上面的文件资源则与 rwx 的权限设置有关。 不过你要注意的是，各种权限设置对 root 是无效的。因此，当某个程序想要对文件进行存取时， 系统就会根据该程序的拥有者/群组，并比对文件的权限，若通过权限检查，就可以存取该文件了。

这种存取文件系统的方式被称为“自主式存取控制 （Discretionary Access Control, DAC）”，基本上，就是依据程序的拥有者与文件资源的 rwx 权限来决定有无存取的能力。 不过这种 DAC 的存取控制有几个困扰，那就是：

- root 具有最高的权限：如果不小心某支程序被有心人士取得， 且该程序属于 root 的权限，那么这支程序就可以在系统上进行任何资源的存取！真是要命！
- 使用者可以取得程序来变更文件资源的存取权限：如果你不小心将某个目录的权限设置为 777 ，由于对任何人的权限会变成 rwx ，因此该目录就会被任何人所任意存取！

这些问题是非常严重的！尤其是当你的系统是被某些漫不经心的系统管理员所掌控时！她们甚至觉得目录权限调为 777 也没有什么了不起的危险哩...

- 以政策规则订定特定程序读取特定文件：委任式存取控制, MAC

现在我们知道 DAC 的困扰就是当使用者取得程序后，他可以借由这支程序与自己默认的权限来处理他自己的文件资源。 万一这个使用者对 Linux 系统不熟，那就很可能会有资源误用的问题产生。为了避免 DAC 容易发生的问题，因此 SELinux 导入了委任式存取控制 （Mandatory Access Control, MAC） 的方法！

委任式存取控制 （MAC） 有趣啦！他可以针对特定的程序与特定的文件资源来进行权限的控管！ 也就是说，即使你是 root ，那么在使用不同的程序时，你所能取得的权限并不一定是 root ， 而得要看当时该程序的设置而定。如此一来，我们针对控制的“主体”变成了“程序”而不是使用者喔！ 此外，这个主体程序也不能任意使用系统文件资源，因为每个文件资源也有针对该主体程序设置可取用的权限！ 如此一来，控制项目就细的多了！但整个系统程序那么多、文件那么多，一项一项控制可就没完没了！ 所以 SELinux 也提供一些默认的政策 （Policy） ，并在该政策内提供多个规则 （rule） ，让你可以选择是否启用该控制规则！

在委任式存取控制的设置下，我们的程序能够活动的空间就变小了！举例来说， WWW 服务器软件的达成程序为 httpd 这支程序， 而默认情况下， httpd 仅能在 /var/www/ 这个目录下面存取文件，如果 httpd 这个程序想要到其他目录去存取数据时， 除了规则设置要开放外，目标目录也得要设置成 httpd 可读取的模式 （type） 才行喔！限制非常多！ 所以，即使不小心 httpd 被 cracker 取得了控制权，他也无权浏览 /etc/shadow 等重要的配置文件喔！

简单的来说，针对 Apache 这个 WWW 网络服务使用 DAC 或 MAC 的结果来说，两者间的关系可以使用下图来说明。 下面这个图示取自 Red Hat 训练教材，真的是很不错～所以被鸟哥借用来说明一下！

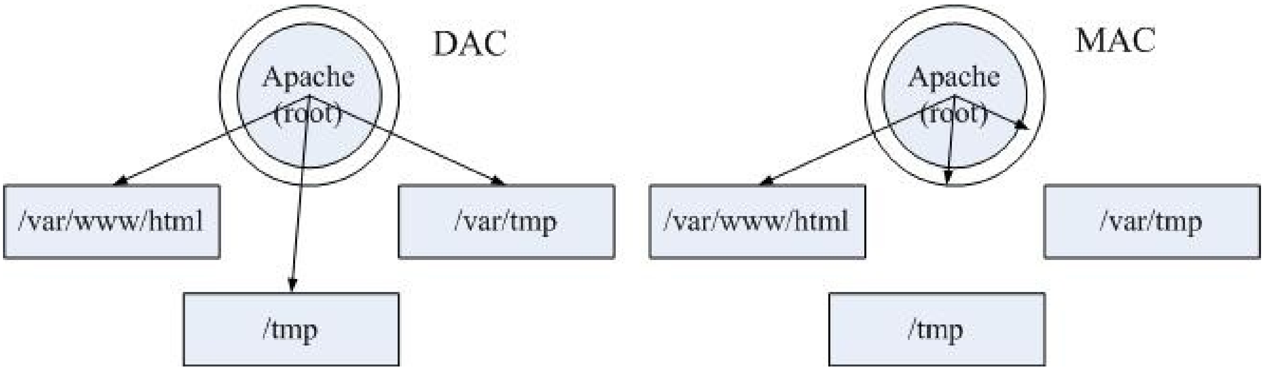


图16.5.1、使用 DAC/MAC 产生的不同结果，以 Apache 为例说明

左图是没有 SELinux 的 DAC 存取结果，apache 这只 root 所主导的程序，可以在这三个目录内作任何文件的新建与修改～ 相当麻烦～右边则是加上 SELinux 的 MAC 管理的结果，SELinux 仅会针对 Apache 这个“ process ”放行部份的目录， 其他的非正规目录就不会放行给 Apache 使用！因此不管你是谁，就是不能穿透 MAC 的框框！这样有比较了解乎？

16.5.2 SELinux 的运行模式

再次的重复说明一下，SELinux 是通过 MAC 的方式来控管程序，他控制的主体是程序， 而目标则是该程序能否读取的“文件资源”！所以先来说明一下这些咚咚的相关性啦！[4]

- 主体 （Subject） ： SELinux 主要想要管理的就是程序，因此你可以将“主体”跟本章谈到的 process 划上等号；
- 目标 （Object） ： 主体程序能否存取的“目标资源”一般就是文件系统。因此这个目标项目可以等文件系统划上等号；
- 政策 （Policy） ： 由于程序与文件数量庞大，因此 SELinux 会依据某些服务来制订基本的存取安全性政策。这些政策内还会有详细的规则 （rule） 来指定不同的服务开放某些资源的存取与否。在目前的 CentOS 7.x 里面仅有提供三个主要的政策，分别是：
  - targeted：针对网络服务限制较多，针对本机限制较少，是默认的政策；
  - minimum：由 target 修订而来，仅针对选择的程序来保护！
  - mls：完整的 SELinux 限制，限制方面较为严格。建议使用默认的 targeted 政策即可。
- 安全性本文 （security context） ： 我们刚刚谈到了主体、目标与政策面，但是主体能不能存取目标除了政策指定之外，主体与目标的安全性本文必须一致才能够顺利存取。 这个安全性本文 （security context） 有点类似文件系统的 rwx 啦！安全性本文的内容与设置是非常重要的！ 如果设置错误，你的某些服务（主体程序）就无法存取文件系统（目标资源），当然就会一直出现“权限不符”的错误讯息了！

由于 SELinux 重点在保护程序读取文件系统的权限，因此我们将上述的几个说明搭配起来，绘制成下面的流程图，比较好理解：

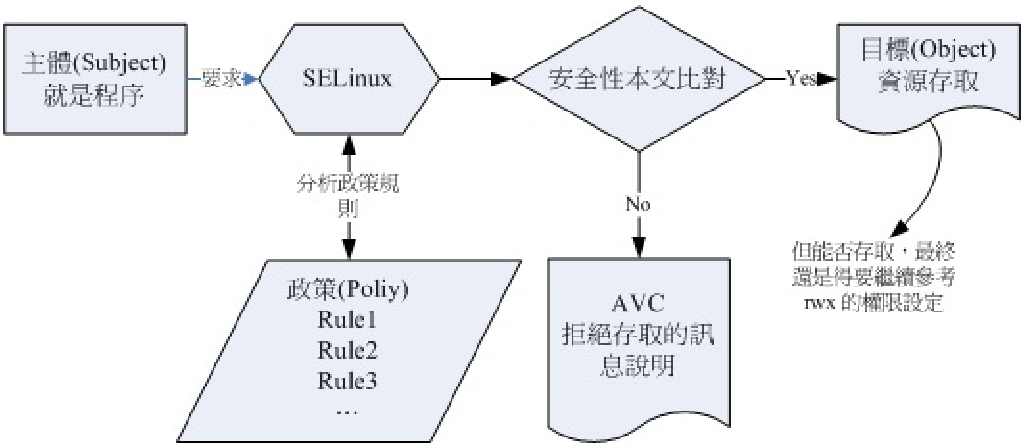


图16.5.2、SELinux 运行的各元件之相关性（本图参考小州老师的上课讲义）

上图的重点在“主体”如何取得“目标”的资源存取权限！ 由上图我们可以发现，（1）主体程序必须要通过 SELinux 政策内的规则放行后，就可以与目标资源进行安全性本文的比对，（2）若比对失败则无法存取目标，若比对成功则可以开始存取目标。问题是，最终能否存取目标还是与文件系统的 rwx 权限设置有关喔！如此一来，加入了 SELinux 之后，出现权限不符的情况时，你就得要一步一步的分析可能的问题了！

- 安全性本文 （Security Context）

CentOS 7.x 的 target 政策已经帮我们制订好非常多的规则了，因此你只要知道如何打开/关闭某项规则的放行与否即可。 那个安全性本文比较麻烦！因为你可能需要自行设置文件的安全性本文呢！为何需要自行设置啊？ 举例来说，你不也常常进行文件的 rwx 的重新设置吗？这个安全性本文你就将他想成 SELinux 内必备的 rwx 就是了！这样比较好理解啦。

安全性本文存在于主体程序中与目标文件资源中。程序在内存内，所以安全性本文可以存入是没问题。 那文件的安全性本文是记录在哪里呢？事实上，安全性本文是放置到文件的 inode 内的，因此主体程序想要读取目标文件资源时，同样需要读取 inode ， 这 inode 内就可以比对安全性本文以及 rwx 等权限值是否正确，而给予适当的读取权限依据。

那么安全性本文到底是什么样的存在呢？我们先来看看 /root 下面的文件的安全性本文好了。 观察安全性本文可使用“ ls -Z ”去观察如下：（注意：你必须已经启动了 SELinux 才行！若尚未启动，这部份请稍微看过一遍即可。下面会介绍如何启动 SELinux 喔！）

```
# 先来观察一下 root 主文件夹下面的“文件的 SELinux 相关信息”
[root@study ~]# ls -Z
-rw-------. root root system_u:object_r:admin_home_t:s0     anaconda-ks.cfg
-rw-r--r--. root root system_u:object_r:admin_home_t:s0     initial-setup-ks.cfg
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 regular_express.txt
# 上述特殊字体的部分，就是安全性本文的内容！鸟哥仅列出数个默认的文件而已，
# 本书学习过程中所写下的文件则没有列在上头喔！
```

如上所示，安全性本文主要用冒号分为三个字段，这三个字段的意义为：

```
Identify:role:type
身份识别:角色:类型
```

这三个字段的意义仔细的说明一下吧：

- 身份识别（Identify）：

相当于帐号方面的身份识别！主要的身份识别常见有下面几种常见的类型：

- unconfined_u：不受限的用户，也就是说，该文件来自于不受限的程序所产生的！一般来说，我们使用可登陆帐号来取得 bash 之后，默认的 bash 环境是不受 SELinux 管制的～因为 bash 并不是什么特别的网络服务！因此，在这个不受 SELinux 所限制的 bash 程序所产生的文件，其身份识别大多就是 unconfined_u 这个“不受限”用户啰！
- system_u：系统用户，大部分就是系统自己产生的文件啰！

基本上，如果是系统或软件本身所提供的文件，大多就是 system_u 这个身份名称，而如果是我们用户通过 bash 自己创建的文件，大多则是不受限的 unconfined_u 身份～如果是网络服务所产生的文件，或者是系统服务运行过程产生的文件，则大部分的识别就会是 system_u 啰！

因为鸟哥这边教大家使用文字界面来产生许多的数据，因此你看上面的三个文件中，系统安装主动产生的 anaconda-ks.cfs 及 initial-setup-ks.cfg 就会是 system_u，而我们自己从网络上面抓下来的 regular_express.txt 就会是 unconfined_u 这个识别啊！

- 角色（Role）：

通过角色字段，我们可以知道这个数据是属于程序、文件资源还是代表使用者。一般的角色有：

- object_r：代表的是文件或目录等文件资源，这应该是最常见的啰；
- system_r：代表的就是程序啦！不过，一般使用者也会被指定成为 system_r 喔！

你也会发现角色的字段最后面使用“ _r ”来结尾！因为是 role 的意思嘛！

- 类型（Type）（最重要！）：

在默认的 targeted 政策中，Identify 与 Role 字段基本上是不重要的！重要的在于这个类型（type）字段！基本上，一个主体程序能不能读取到这个文件资源，与类型字段有关！而类型字段在文件与程序的定义不太相同，分别是：

- type：在文件资源（Object）上面称为类型（Type）；
- domain：在主体程序（Subject）则称为领域（domain）了！

domain 需要与 type 搭配，则该程序才能够顺利的读取文件资源啦！

- 程序与文件 SELinux type 字段的相关性

那么这三个字段如何利用呢？首先我们来瞧瞧主体程序在这三个字段的意义为何！通过身份识别与角色字段的定义，我们可以约略知道某个程序所代表的意义喔！先来动手瞧一瞧目前系统中的程序在 SELinux 下面的安全本文为何？

```
# 再来观察一下系统“程序的 SELinux 相关信息”
[root@study ~]# ps -eZ
LABEL                             PID TTY          TIME CMD
system_u:system_r:init_t:s0         1 ?        00:00:03 systemd
system_u:system_r:kernel_t:s0       2 ?        00:00:00 kthreadd
system_u:system_r:kernel_t:s0       3 ?        00:00:00 ksoftirqd/0
.....（中间省略）.....
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 31513 ? 00:00:00 sshd
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 31535 pts/0 00:00:00 bash
# 基本上程序主要就分为两大类，一种是系统有受限的 system_u:system_r，另一种则可能是用户自己的，
# 比较不受限的程序 （通常是本机用户自己执行的程序），亦即是 unconfined_u:unconfined_r 这两种！
```

基本上，这些对应数据在 targeted 政策下的对应如下：

身份识别	角色	该对应在 **targeted** 的意义
unconfined_u	unconfined_r	一般可登陆使用者的程序啰！比较没有受限的程序之意！大多数都是用户已经顺利登陆系统 （不论是网络还是本机登陆来取得可用的 shell） 后， 所用来操作系统的程序！如 bash, X window 相关软件等。
system_u	system_r	由于为系统帐号，因此是非交谈式的系统运行程序，大多数的系统程序均是这种类型！

但就如上所述，在默认的 target 政策下，其实最重要的字段是类型字段 （type）， 主体与目标之间是否具有可以读写的权限，与程序的 domain 及文件的 type 有关！这两者的关系我们可以使用 crond 以及他的配置文件来说明！ 亦即是 /usr/sbin/crond, /etc/crontab, /etc/cron.d 等文件来说明。 首先，看看这几个咚咚的安全性本文内容先：

```
# 1\. 先看看 crond 这个“程序”的安全本文内容：
[root@study ~]# ps -eZ &#124; grep cron
system_u:system_r:crond_t:s0-s0:c0.c1023 1338 ? 00:00:01 crond
system_u:system_r:crond_t:s0-s0:c0.c1023 1340 ? 00:00:00 atd
# 这个安全本文的类型名称为 crond_t 格式！

# 2\. 再来瞧瞧可执行文件、配置文件等等的安全本文内容为何！
[root@study ~]# ll -Zd /usr/sbin/crond /etc/crontab /etc/cron.d
drwxr-xr-x. root root system_u:object_r:system_cron_spool_t:s0 /etc/cron.d
-rw-r--r--. root root system_u:object_r:system_cron_spool_t:s0 /etc/crontab
-rwxr-xr-x. root root system_u:object_r:crond_exec_t:s0 /usr/sbin/crond
```

当我们执行 /usr/sbin/crond 之后，这个程序变成的程序的 domain 类型会是 crond_t 这一个～而这个 crond_t 能够读取的配置文件则为 system_cron_spool_t 这种的类型。因此不论 /etc/crontab, /etc/cron.d 以及 /var/spool/cron 都会是相关的 SELinux 类型 （/var/spool/cron 为 user_cron_spool_t）。 文字看起来不太容易了解，我们使用图示来说明这几个东西的关系！

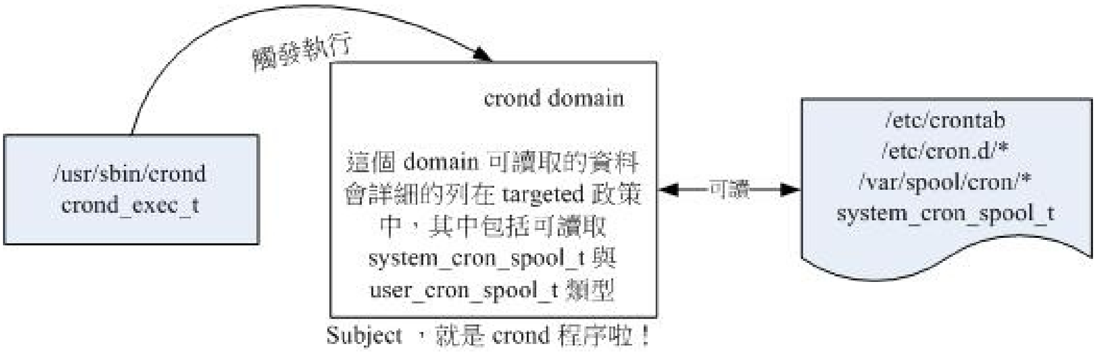


图16.5.3、主体程序取得的 domain 与目标文件资源的 type 相互关系以 crond 为例

上图的意义我们可以这样看的：

1. 首先，我们触发一个可执行的目标文件，那就是具有 crond_exec_t 这个类型的 /usr/sbin/crond 文件；
2. 该文件的类型会让这个文件所造成的主体程序 （Subject） 具有 crond 这个领域 （domain）， 我们的政策针对这个领域已经制定了许多规则，其中包括这个领域可以读取的目标资源类型；
3. 由于 crond domain 被设置为可以读取 system_cron_spool_t 这个类型的目标文件 （Object）， 因此你的配置文件放到 /etc/cron.d/ 目录下，就能够被 crond 那支程序所读取了；
4. 但最终能不能读到正确的数据，还得要看 rwx 是否符合 Linux 权限的规范！

上述的流程告诉我们几个重点，第一个是政策内需要制订详细的 domain/type 相关性；第二个是若文件的 type 设置错误， 那么即使权限设置为 rwx 全开的 777 ，该主体程序也无法读取目标文件资源的啦！不过如此一来， 也就可以避免使用者将他的主文件夹设置为 777 时所造成的权限困扰。

眞的是这样吗？没关系～让我们来做个测试练习吧！就是，万一你的 crond 配置文件的 SELinux 并不是 system_cron_spool_t 时， 该配置文件眞的可以顺利的被读取运行吗？来看看下面的范例！

```
# 1\. 先假设你因为不熟的缘故，因此是在“root 主文件夹”创建一个如下的 cron 设置：
[root@study ~]# vim checktime
10 * * * * root sleep 60s

# 2\. 检查后才发现文件放错目录了，又不想要保留副本，因此使用 mv 移动到正确目录：
[root@study ~]# mv checktime /etc/cron.d
[root@study ~]# ll /etc/cron.d/checktime
-rw-r--r--. 1 root root 27 Aug  7 18:41 /etc/cron.d/checktime
# 仔细看喔，权限是 644 ，确定没有问题！任何程序都能够读取喔！

# 3\. 强制重新启动 crond ，然后偷看一下登录文件，看看有没有问题发生！
[root@study ~]# systemctl restart crond
[root@study ~]# tail /var/log/cron
Aug  7 18:46:01 study crond[28174]: （（null）） Unauthorized SELinux context=system_u:sys
system_cronjob_t:s0-s0:c0.c1023 file_context=unconfined_u:object_r:admin_home_t:s0
（/etc/cron.d/checktime）
Aug  7 18:46:01 study crond[28174]: （root） FAILED （loading cron table）
# 上面的意思是，有错误！因为原本的安全本文与文件的实际安全本文无法搭配的缘故！
```

您瞧瞧～从上面的测试案例来看，我们的配置文件确实没有办法被 crond 这个服务所读取喔！而原因在登录文件内就有说明， 主要就是来自 SELinux 安全本文 （context） type 的不同所致喔！没办法读就没办法读，先放着～后面再来学怎么处理这问题吧！

16.5.3 SELinux 三种模式的启动、关闭与观察

并非所有的 Linux distributions 都支持 SELinux 的，所以你必须要先观察一下你的系统版本为何！ 鸟哥这里介绍的 CentOS 7.x 本身就有支持 SELinux 啦！所以你不需要自行编译 SELinux 到你的 Linux 核心中！ 目前 SELinux 依据启动与否，共有三种模式，分别如下：

- enforcing：强制模式，代表 SELinux 运行中，且已经正确的开始限制 domain/type 了；
- permissive：宽容模式：代表 SELinux 运行中，不过仅会有警告讯息并不会实际限制 domain/type 的存取。这种模式可以运来作为 SELinux 的 debug 之用；
- disabled：关闭，SELinux 并没有实际运行。

这三种模式跟图16.5.2之间的关系如何呢？我们前面不是谈过主体程序需要经过政策规则、安全本文比对之后，加上 rwx 的权限规范， 若一切合理才会让程序顺利的读取文件吗？那么这个 SELinux 的三种模式与上面谈到的政策规则、安全本文的关系为何呢？我们还是使用图示加上流程来让大家理解一下：

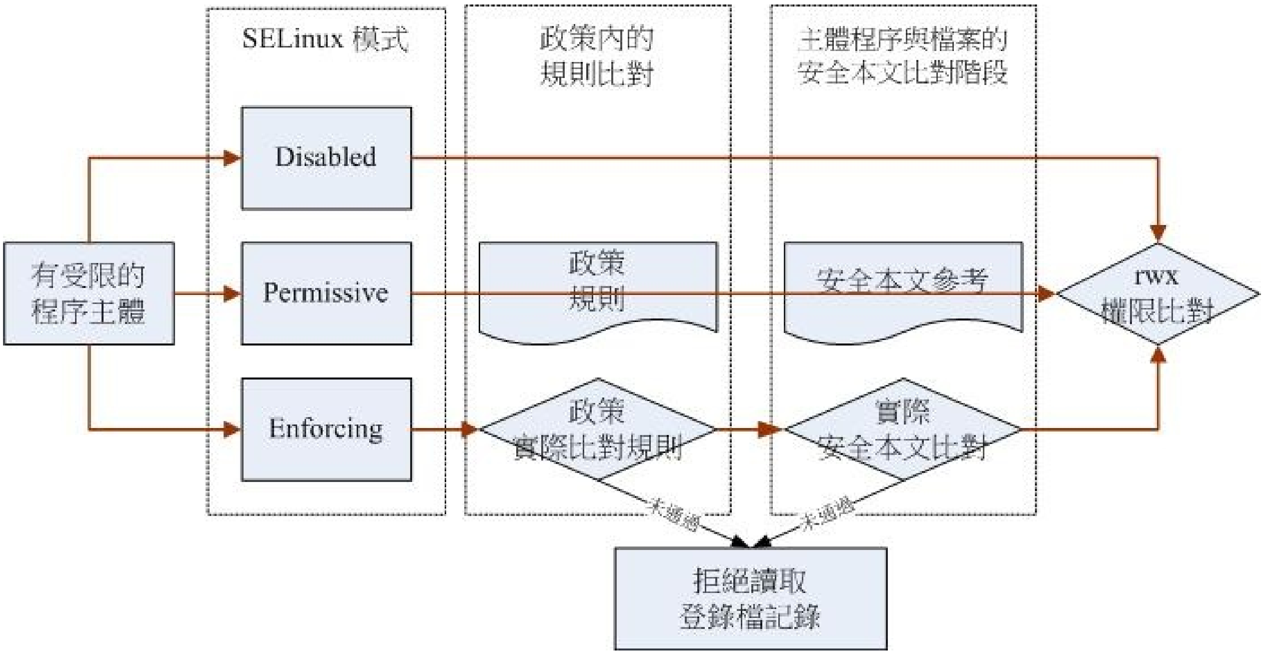


图16.5.4、SELinux 的三种类型与实际运行流程图示意

就如上图所示，首先，你得要知道，并不是所有的程序都会被 SELinux 所管制，因此最左边会出现一个所谓的“有受限的程序主体”！那如何观察有没有受限 （confined ）呢？ 很简单啊！就通过 ps -eZ 去撷取！举例来说，我们来找一找 crond 与 bash 这两只程序是否有被限制吧？

```
[root@study ~]# ps -eZ &#124; grep -E 'cron&#124;bash'
system_u:system_r:crond_t:s0-s0:c0.c1023 1340 ? 00:00:00 atd
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 13888 tty2 00:00:00 bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 28054 pts/0 00:00:00 bash
unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023 28094 pts/0 00:00:00 bash
system_u:system_r:crond_t:s0-s0:c0.c1023 28174 ? 00:00:00 crond
```

如前所述，因为在目前 target 这个政策下面，只有第三个类型 （type） 字段会有影响，因此我们上表仅列出第三个字段的数据而已。 我们可以看到， crond 确实是有受限的主体程序，而 bash 因为是本机程序，因此就是不受限 （unconfined_t） 的类型！也就是说， bash 是不需要经过图 16.5.4 的流程，而是直接去判断 rwx 而已～。

了解了受限的主体程序的意义之后，再来了解一下，三种模式的运行吧！首先，如果是 Disabled 的模式，那么 SELinux 将不会运行，当然受限的程序也不会经过 SELinux ， 也是直接去判断 rwx 而已。那如果是宽容 （permissive） 模式呢？这种模式也是不会将主体程序抵挡 （所以箭头是可以直接穿透的喔！），不过万一没有通过政策规则，或者是安全本文的比对时， 那么该读写动作将会被纪录起来 （log），可作为未来检查问题的判断依据。

至于最终那个 Enforcing 模式，就是实际将受限主体进入规则比对、安全本文比对的流程，若失败，就直接抵挡主体程序的读写行为，并且将他记录下来。 如果通通没问题，这才进入到 rwx 权限的判断喔！这样可以理解三种模式的行为了吗？

那你怎么知道目前的 SELinux 模式呢？就通过 getenforce 吧！

```
[root@study ~]# getenforce
Enforcing  &lt;==诺！就显示出目前的模式为 Enforcing 啰！
```

另外，我们又如何知道 SELinux 的政策 （Policy） 为何呢？这时可以使用 sestatus 来观察：

```
[root@study ~]# sestatus [-vb]
选项与参数：
-v  ：检查列于 /etc/sestatus.conf 内的文件与程序的安全性本文内容；
-b  ：将目前政策的规则布林值列出，亦即某些规则 （rule） 是否要启动 （0/1） 之意；

范例一：列出目前的 SELinux 使用哪个政策 （Policy）？
[root@study ~]# sestatus
SELinux status:                 enabled           &lt;==是否启动 SELinux
SELinuxfs mount:                /sys/fs/selinux   &lt;==SELinux 的相关文件数据挂载点
SELinux root directory:         /etc/selinux      &lt;==SELinux 的根目录所在
Loaded policy name:             targeted          &lt;==目前的政策为何？
Current mode:                   enforcing         &lt;==目前的模式
Mode from config file:          enforcing         &lt;==目前配置文件内规范的 SELinux 模式
Policy MLS status:              enabled           &lt;==是否含有 MLS 的模式机制
Policy deny_unknown status:     allowed           &lt;==是否默认抵挡未知的主体程序
Max kernel policy version:      28
```

如上所示，目前是启动的，而且是 Enforcing 模式，而由配置文件查询得知亦为 Enforcing 模式。 此外，目前的默认政策为 targeted 这一个。你应该要有疑问的是， SELinux 的配置文件是哪个文件啊？ 其实就是 /etc/selinux/config 这个文件喔！我们来看看内容：

```
[root@study ~]# vim /etc/selinux/config
SELINUX=enforcing     &lt;==调整 enforcing&#124;disabled&#124;permissive
SELINUXTYPE=targeted  &lt;==目前仅有 targeted, mls, minimum 三种政策
```

若有需要修改默认政策的话，就直接改 SELINUX=enforcing 那一行即可喔！

- SELinux 的启动与关闭

上面是默认的政策与启动的模式！你要注意的是，如果改变了政策则需要重新开机；如果由 enforcing 或 permissive 改成 disabled ，或由 disabled 改成其他两个，那也必须要重新开机。这是因为 SELinux 是整合到核心里面去的， 你只可以在 SELinux 运行下切换成为强制（enforcing） 或宽容 （permissive） 模式，不能够直接关闭 SELinux 的！ 如果刚刚你发现 getenforce 出现 disabled 时，请到上述文件修改成为 enforcing 然后重新开机吧！

不过你要注意的是，如果从 disable 转到启动 SELinux 的模式时， 由于系统必须要针对文件写入安全性本文的信息，因此开机过程会花费不少时间在等待重新写入 SELinux 安全性本文 （有时也称为 SELinux Label） ，而且在写完之后还得要再次的重新开机一次喔！你必须要等待粉长一段时间！ 等到下次开机成功后，再使用 getenforce 或 sestatus 来观察看看有否成功的启动到 Enforcing 的模式啰！

如果你已经在 Enforcing 的模式，但是可能由于一些设置的问题导致 SELinux 让某些服务无法正常的运行， 此时你可以将 Enforcing 的模式改为宽容 （permissive） 的模式，让 SELinux 只会警告无法顺利连线的讯息， 而不是直接抵挡主体程序的读取权限。让 SELinux 模式在 enforcing 与 permissive 之间切换的方法为：

```
[root@study ~]# setenforce [0&#124;1]
选项与参数：
0 ：转成 permissive 宽容模式；
1 ：转成 Enforcing 强制模式

范例一：将 SELinux 在 Enforcing 与 permissive 之间切换与观察
[root@study ~]# setenforce 0
[root@study ~]# getenforce
Permissive
[root@study ~]# setenforce 1
[root@study ~]# getenforce
Enforcing
```

不过请注意， setenforce 无法在 Disabled 的模式下面进行模式的切换喔！



Tips 在某些特殊的情况下面，你从 Disabled 切换成 Enforcing 之后，竟然有一堆服务无法顺利启动，都会跟你说在 /lib/xxx 里面的数据没有权限读取，所以启动失败。这大多是由于在重新写入 SELinux type （Relabel） 出错之故，使用 Permissive 就没有这个错误。那如何处理

呢？最简单的方法就是在 Permissive 的状态下，使用“ restorecon -Rv / ”重新还原所有 SELinux 的类型，就能够处理这个错误！

16.5.4 SELinux 政策内的规则管理

从图 16.5.4 里面，我们知道 SELinux 的三种模式是会影响到主体程序的放行与否。 如果是进入 Enforcing 模式，那么接着下来会影响到主体程序的，当然就是第二关：“ target 政策内的各项规则 （rules） ”了！ 好了，那么我们怎么知道目前这个政策里面到底有多少会影响到主体程序的规则呢？很简单，就通过 getsebool 来瞧一瞧即可。

- SELinux 各个规则的布林值查询 getsebool

如果想要查询系统上面全部规则的启动与否 （on/off，亦即布林值） ，很简单的通过 sestatus -b 或 getsebool -a 均可！

```
[root@study ~]# getsebool [-a] [规则的名称]
选项与参数：
-a  ：列出目前系统上面的所有 SELinux 规则的布林值为打开或关闭值

范例一：查询本系统内所有的布林值设置状况
[root@study ~]# getsebool -a
abrt_anon_write --&gt; off
abrt_handle_event --&gt; off
....（中间省略）....
cron_can_relabel --&gt; off                # 这个跟 cornd 比较有关！
cron_userdomain_transition --&gt; on
....（中间省略）....
httpd_enable_homedirs --&gt; off           # 这当然就是跟网页，亦即 http 有关的啰！
....（下面省略）....
# 这么多的 SELinux 规则喔！每个规则后面都列出现在是允许放行还是不许放行的布林值喔！
```

- SELinux 各个规则规范的主体程序能够读取的文件 SELinux type 查询 seinfo, sesearch

我们现在知道有这么多的 SELinux 规则，但是每个规则内到底是在限制什么东西？如果你想要知道的话，那就得要使用 seinfo 等工具！ 这些工具并没有在我们安装时就安装了，因此请拿出原版光盘，放到光驱，鸟哥假设你将原版光盘挂载到 /mnt 下面，那么接下来这么作， 先安装好我们所需要的软件才行！

```
[root@study ~]# yum install /mnt/Packages/setools-console-*
```

很快的安装完毕之后，我们就可以来使用 seinfo, sesearch 等指令了！

```
[root@study ~]# seinfo [-Atrub]
选项与参数：
-A  ：列出 SELinux 的状态、规则布林值、身份识别、角色、类别等所有信息
-u  ：列出 SELinux 的所有身份识别 （user） 种类
-r  ：列出 SELinux 的所有角色 （role） 种类
-t  ：列出 SELinux 的所有类别 （type） 种类
-b  ：列出所有规则的种类 （布林值）

范例一：列出 SELinux 在此政策下的统计状态
[root@study ~]# seinfo
Statistics for policy file: /sys/fs/selinux/policy
Policy Version & Type: v.28 （binary, mls）

   Classes:             83    Permissions:       255
   Sensitivities:        1    Categories:       1024
   Types:             4620    Attributes:        357
   Users:                8    Roles:              14
   Booleans:           295    Cond. Expr.:       346
   Allow:           102249    Neverallow:          0
   Auditallow:         160    Dontaudit:        8413
   Type_trans:       16863    Type_change:        74
   Type_member:         35    Role allow:         30
   Role_trans:         412    Range_trans:      5439
....（下面省略）....
# 从上面我们可以看到这个政策是 targeted ，此政策的安全本文类别有 4620 个；
# 而各种 SELinux 的规则 （Booleans） 共制订了 295 条！
```

我们在 16.5.2 里面简单的谈到了几个身份识别 （user） 以及角色 （role） 而已，如果你想要查询目前所有的身份识别与角色，就使用“ seinfo -u ”及“ seinfo -r ”就可以知道了！至于简单的统计数据，就直接输入 seinfo 即可！但是上面还是没有谈到规则相关的东西耶～ 没关系～一个一个来～我们在 16.5.1 的最后面谈到 /etc/cron.d/checktime 的 SELinux type 类型不太对～那我们也知道 crond 这个程序的 type 是 crond_t ， 能不能找一下 crond_t 能够读取的文件 SELinux type 有哪些呢？

```
[root@study ~]# sesearch [-A] [-s 主体类别] [-t 目标类别] [-b 布林值]
选项与参数：
-A  ：列出后面数据中，允许“读取或放行”的相关数据
-t  ：后面还要接类别，例如 -t httpd_t
-b  ：后面还要接SELinux的规则，例如 -b httpd_enable_ftp_server

范例一：找出 crond_t 这个主体程序能够读取的文件 SELinux type
[root@study ~]# sesearch -A -s crond_t &#124; grep spool
   allow crond_t system_cron_spool_t : file { ioctl read write create getattr ..
   allow crond_t system_cron_spool_t : dir { ioctl read getattr lock search op..
   allow crond_t user_cron_spool_t : file { ioctl read write create getattr se..
   allow crond_t user_cron_spool_t : dir { ioctl read write getattr lock add_n..
   allow crond_t user_cron_spool_t : lnk_file { read getattr } ;
# allow 后面接主体程序以及文件的 SELinux type，上面的数据是撷取出来的，
# 意思是说，crond_t 可以读取 system_cron_spool_t 的文件/目录类型～等等！

范例二：找出 crond_t 是否能够读取 /etc/cron.d/checktime 这个我们自订的配置文件？
[root@study ~]# ll -Z /etc/cron.d/checktime
-rw-r--r--. root root unconfined_u:object_r:admin_home_t:s0 /etc/cron.d/checktime
# 两个重点，一个是 SELinux type 为 admin_home_t，一个是文件 （file）

[root@study ~]# sesearch -A -s crond_t &#124; grep admin_home_t
   allow domain admin_home_t : dir { getattr search open } ;
   allow domain admin_home_t : lnk_file { read getattr } ;
   allow crond_t admin_home_t : dir { ioctl read getattr lock search open } ;
   allow crond_t admin_home_t : lnk_file { read getattr } ;
# 仔细看！看仔细～虽然有 crond_t admin_home_t 存在，但是这是总体的信息，
# 并没有针对某些规则的寻找～所以还是不确定 checktime 能否被读取。但是，基本上就是 SELinux
# type 出问题～因此才会无法读取的！
```

所以，现在我们知道 /etc/cron.d/checktime 这个我们自己复制过去的文件会没有办法被读取的原因，就是因为 SELinux type 错误啦！ 根本就无法被读取～好～那现在我们来查一查，那 getsebool -a 里面看到的 httpd_enable_homedirs 到底是什么？又是规范了哪些主体程序能够读取的 SELinux type 呢？

```
[root@study ~]# semanage boolean -l &#124; grep httpd_enable_homedirs
SELinux boolean                State  Default Description
httpd_enable_homedirs           (off  ,  off)   Allow httpd to enable homedirs
# httpd_enable_homedirs 的功能是允许 httpd 程序去读取使用者主文件夹的意思～

[root@study ~]# sesearch -A -b httpd_enable_homedirs
范例三：列出 httpd_enable_homedirs 这个规则当中，主体程序能够读取的文件 SELinux type
Found 43 semantic av rules:
   allow httpd_t home_root_t : dir { ioctl read getattr lock search open } ;
   allow httpd_t home_root_t : lnk_file { read getattr } ;
   allow httpd_t user_home_type : dir { getattr search open } ;
   allow httpd_t user_home_type : lnk_file { read getattr } ;
....（后面省略）....
# 从上面的数据才可以理解，在这个规则中，主要是放行 httpd_t 能否读取使用者主文件夹的文件！
# 所以，如果这个规则没有启动，基本上， httpd_t 这种程序就无法读取使用者主文件夹下的文件！
```

- 修改 SELinux 规则的布林值 setsebool

那么如果查询到某个 SELinux rule，并且以 sesearch 知道该规则的用途后，想要关闭或启动他，又该如何处置？

```
[root@study ~]# setsebool  [-P]  “规则名称” [0&#124;1]
选项与参数：
-P  ：直接将设置值写入配置文件，该设置数据未来会生效的！

范例一：查询 httpd_enable_homedirs 这个规则的状态，并且修改这个规则成为不同的布林值
[root@study ~]# getsebool httpd_enable_homedirs
httpd_enable_homedirs --&gt; off  &lt;==结果是 off ，依题意给他启动看看！

[root@study ~]# setsebool -P httpd_enable_homedirs 1 # 会跑很久很久！请耐心等待！
[root@study ~]# getsebool httpd_enable_homedirs
httpd_enable_homedirs --&gt; on
```

这个 setsebool 最好记得一定要加上 -P 的选项！因为这样才能将此设置写入配置文件！ 这是非常棒的工具组！你一定要知道如何使用 getsebool 与 setsebool 才行！

16.5.5 SELinux 安全本文的修改

再次的回到图 16.5.4 上头去，现在我们知道 SELinux 对受限的主体程序有没有影响，第一关考虑 SELinux 的三种类型，第二关考虑 SELinux 的政策规则是否放行，第三关则是开始比对 SELinux type 啦！从刚刚 16.5.4 小节我们也知道可以通过 sesearch 来找到主体程序与文件的 SELinux type 关系！ 好，现在总算要来修改文件的 SELinux type，以让主体程序能够读到正确的文件啊！这时就得要几个重要的小东西了～来瞧瞧～

- 使用 chcon 手动修改文件的 SELinux type

```
[root@study ~]# chcon [-R] [-t type] [-u user] [-r role] 文件
[root@study ~]# chcon [-R] --reference=范例档 文件
选项与参数：
-R  ：连同该目录下的次目录也同时修改；
-t  ：后面接安全性本文的类型字段！例如 httpd_sys_content_t ；
-u  ：后面接身份识别，例如 system_u； （不重要）
-r  ：后面街角色，例如 system_r；     （不重要）
-v  ：若有变化成功，请将变动的结果列出来
--reference=范例档：拿某个文件当范例来修改后续接的文件的类型！

范例一：查询一下 /etc/hosts 的 SELinux type，并将该类型套用到 /etc/cron.d/checktime 上
[root@study ~]# ll -Z /etc/hosts
-rw-r--r--. root root system_u:object_r:net_conf_t:s0  /etc/hosts
[root@study ~]# chcon -v -t net_conf_t /etc/cron.d/checktime
changing security context of '/etc/cron.d/checktime'
[root@study ~]# ll -Z /etc/cron.d/checktime
-rw-r--r--. root root unconfined_u:object_r:net_conf_t:s0 /etc/cron.d/checktime

范例二：直接以 /etc/shadow SELinux type 套用到 /etc/cron.d/checktime 上！
[root@study ~]# chcon -v --reference=/etc/shadow /etc/cron.d/checktime
[root@study ~]# ll -Z /etc/shadow /etc/cron.d/checktime
-rw-r--r--. root root system_u:object_r:shadow_t:s0    /etc/cron.d/checktime
----------. root root system_u:object_r:shadow_t:s0    /etc/shadow
```

上面的练习“都没有正确的解答！”因为正确的 SELinux type 应该就是要以 /etc/cron.d/ 下面的文件为标准来处理才对啊～ 好了～既然如此～能不能让 SELinux 自己解决默认目录下的 SELinux type 呢？可以！就用 restorecon 吧！

- 使用 restorecon 让文件恢复正确的 SELinux type

```
[root@study ~]# restorecon [-Rv] 文件或目录
选项与参数：
-R  ：连同次目录一起修改；
-v  ：将过程显示到屏幕上

范例三：将 /etc/cron.d/ 下面的文件通通恢复成默认的 SELinux type！
[root@study ~]# restorecon -Rv /etc/cron.d
restorecon reset /etc/cron.d/checktime context system_u:object_r:shadow_t:s0-&gt;
system_u:object_r:system_cron_spool_t:s0
# 上面这两行其实是同一行喔！表示将 checktime 由 shadow_t 改为 system_cron_spool_t

范例四：重新启动 crond 看看有没有正确启动 checktime 啰！？
[root@study ~]# systemctl restart crond
[root@study ~]# tail /var/log/cron
# 再去瞧瞧这个 /var/log/cron 的内容，应该就没有错误讯息了
```

其实，鸟哥几乎已经忘了 chcon 这个指令了！因为 restorecon 主动的回复默认的 SELinux type 要简单很多！而且可以一口气恢复整个目录下的文件！ 所以，鸟哥建议你几乎只要记得 restorecon 搭配 -Rv 同时加上某个目录这样的指令串即可～修改 SELinux 的 type 就变得非常的轻松啰！

- semanage 默认目录的安全性本文查询与修改

你应该要觉得奇怪，为什么 restorecon 可以“恢复”原本的 SELinux type 呢？那肯定就是有个地方在纪录每个文件/目录的 SELinux 默认类型啰？ 没错！是这样～那要如何 （1）查询默认的 SELinux type 以及 （2）如何增加/修改/删除默认的 SELinux type 呢？很简单～通过 semanage 即可！他是这样使用的：

```
[root@study ~]# semanage {login&#124;user&#124;port&#124;interface&#124;fcontext&#124;tra
[root@study ~]# semanage fcontext -{a&#124;d&#124;m} [-frst] file_spec
选项与参数：
fcontext ：主要用在安全性本文方面的用途， -l 为查询的意思；
-a ：增加的意思，你可以增加一些目录的默认安全性本文类型设置；
-m ：修改的意思；
-d ：删除的意思。

范例一：查询一下 /etc /etc/cron.d 的默认 SELinux type 为何？
[root@study ~]# semanage fcontext -l &#124; grep -E '^/etc &#124;^/etc/cron'
SELinux fcontext         type               Context
/etc                     all files          system_u:object_r:etc_t:s0
/etc/cron\.d（/.*）?       all files          system_u:object_r:system_cron_spool_t:s0
```

看到上面输出的最后一行，那也是为啥我们直接使用 vim 去 /etc/cron.d 下面创建新文件时，默认的 SELinux type 就是正确的！ 同时，我们也会知道使用 restorecon 回复正确的 SELinux type 时，系统会去判断默认的类型为何的依据。现在让我们来想一想， 如果 （当然是假的！不可能这么干） 我们要创建一个 /srv/mycron 的目录，这个目录默认也是需要变成 system_cron_spool_t 时， 我们应该要如何处理呢？基本上可以这样作：

```
# 1\. 先创建 /srv/mycron 同时在内部放入配置文件，同时观察 SELinux type
[root@study ~]# mkdir /srv/mycron
[root@study ~]# cp /etc/cron.d/checktime /srv/mycron
[root@study ~]# ll -dZ /srv/mycron /srv/mycron/checktime
drwxr-xr-x. root root unconfined_u:object_r:var_t:s0   /srv/mycron
-rw-r--r--. root root unconfined_u:object_r:var_t:s0   /srv/mycron/checktime

# 2\. 观察一下上层 /srv 的 SELinux type
[root@study ~]# semanage fcontext -l &#124; grep '^/srv'
SELinux fcontext         type               Context
/srv                     all files          system_u:object_r:var_t:s0
# 怪不得 mycron 会是 var_t 啰！

# 3\. 将 mycron 默认值改为 system_cron_spool_t 啰！
[root@study ~]# semanage fcontext -a -t system_cron_spool_t "/srv/mycron（/.*）?"
[root@study ~]# semanage fcontext -l &#124; grep '^/srv/mycron'
SELinux fcontext         type               Context
/srv/mycron（/.*）?        all files          system_u:object_r:system_cron_spool_t:s0

# 4\. 恢复 /srv/mycron 以及子目录相关的 SELinux type 喔！
[root@study ~]# restorecon -Rv /srv/mycron
[root@study ~]# ll -dZ /srv/mycron /srv/mycron/*
drwxr-xr-x. root root unconfined_u:object_r:system_cron_spool_t:s0 /srv/mycron
-rw-r--r--. root root unconfined_u:object_r:system_cron_spool_t:s0 /srv/mycron/checktime
# 有了默认值，未来就不会不小心被乱改了！这样比较妥当些～
```

semanage 的功能很多，不过鸟哥主要用到的仅有 fcontext 这个项目的动作而已。如上所示， 你可以使用 semanage 来查询所有的目录默认值，也能够使用他来增加默认值的设置！ 如果您学会这些基础的工具， 那么 SELinux 对你来说，也不是什么太难的咚咚啰！

16.5.6 一个网络服务案例及登录文件协助

本章在 SELinux 小节当中谈到的各个指令中，尤其是 setsebool, chcon, restorecon 等，都是为了当你的某些网络服务无法正常提供相关功能时， 才需要进行修改的一些指令动作。但是，我们怎么知道哪个时候才需要进行这些指令的修改啊？我们怎么知道系统因为 SELinux 的问题导致网络服务不对劲啊？如果都要靠用户端连线失败才来哭诉，那也太没有效率了！所以，我们的 CentOS 7.x 有提供几支侦测的服务在登录 SELinux 产生的错误喔！那就是 auditd 与 setroubleshootd。

- setroubleshoot --> 错误讯息写入 /var/log/messages

几乎所有 SELinux 相关的程序都会以 se 为开头，这个服务也是以 se 为开头！而 troubleshoot 大家都知道是错误克服，因此这个 setroubleshoot 自然就得要启动他啦！这个服务会将关于 SELinux 的错误讯息与克服方法记录到 /var/log/messages 与 /var/log/setroubleshoot/* 里头，所以你一定得要启动这个服务才好。启动这个服务之前当然就是得要安装它啦！ 这玩意儿总共需要两个软件，分别是 setroublshoot 与 setroubleshoot-server，如果你没有安装，请自行使用 yum 安装吧！

此外，原本的 SELinux 信息本来是以两个服务来记录的，分别是 auditd 与 setroubleshootd。既然是同样的信息，因此 CentOS 6.x （含 7.x） 以后将两者整合在 auditd 当中啦！所以，并没有 setroubleshootd 的服务存在了喔！因此，当你安装好了 setroubleshoot-server 之后，请记得要重新启动 auditd，否则 setroubleshootd 的功能不会被启动的。



Tips 事实上，CentOS 7.x 对 setroubleshootd 的运行方式是： （1）先由 auditd 去调用 audispd 服务， （2）然后 audispd 服务去启动 sedispatch 程序， （3）sedispatch 再将原本的 auditd 讯息转成 setroubleshootd 的讯息，进一步储存下来的！

```
[root@study ~]# rpm -qa &#124; grep setroubleshoot
setroubleshoot-plugins-3.0.59-1.el7.noarch
setroubleshoot-3.2.17-3.el7.x86_64
setroubleshoot-server-3.2.17-3.el7.x86_64
```

在默认的情况下，这个 setroubleshoot 应该都是会安装的！是否正确安装可以使用上述的表格指令去查询。万一没有安装，请使用 yum install 去安装吧！ 再说一遍，安装完毕最好重新启动 auditd 这个服务喔！不过，刚刚装好且顺利启动后， setroubleshoot 还是不会有作用，为啥？ 因为我们并没有任何受限的网络服务主体程序在运行啊！所以，下面我们将使用一个简单的 FTP 服务器软件为例，让你了解到我们上头讲到的许多重点的应用！

- 实例状况说明：通过 vsftpd 这个 FTP 服务器来存取系统上的文件

现在的年轻小伙子们传数据都用 line, FB, dropbox, google 云端磁盘等等，不过在网络早期传送大容量的文件，还是以 FTP 这个协定为主！ 现在为了速度，经常有 p2p 的软件提供大容量文件的传输，但以鸟哥这个老人家来说，可能 FTP 传送数据还是比较有保障... 在 CentOS 7.x 的环境下，达成 FTP 的默认服务器软件主要是 vsftpd 这一支喔！

详细的 FTP 协定我们在服务器篇再来谈，这里只是简单的利用 vsftpd 这个软件与 FTP 的协定来讲解 SELinux 的问题与错误克服而已。 不过既然要使用到 FTP 协定，一些简单的知识还是得要存在才好！否则等一下我们没有办法了解为啥要这么做！ 首先，你得要知道，用户端需要使用“FTP 帐号登陆 FTP 服务器”才行！而有一个称为“匿名 （anonymous） ”的帐号可以登陆系统！ 但是这个匿名的帐号登陆后，只能存取某一个特定的目录，而无法脱离该目录～！

在 vsftpd 中，一般用户与匿名者的主文件夹说明如下：

- 匿名者：如果使用浏览器来连线到 FTP 服务器的话，那默认就是使用匿名者登陆系统。 而匿名者的主文件夹默认是在 /var/ftp 当中！ 同时，匿名者在主文件夹下只能下载数据，不能上传数据到 FTP 服务器。同时，匿名者无法离开 FTP 服务器的 /var/ftp 目录喔！
- 一般 FTP 帐号：在默认的情况下，所有 UID 大于 1000 的帐号，都可以使用 FTP 来登陆系统！ 而登陆系统之后，所有的帐号都能够取得自己主文件夹下面的文件数据！当然默认是可以上传、下载文件的！

为了避免跟之前章节的用户产生误解的情况，这里我们先创建一个名为 ftptest 的帐号，且帐号密码为 myftp123， 先来创建一下吧！

```
[root@study ~]# useradd -s /sbin/nologin ftptest
[root@study ~]# echo "myftp123" &#124; passwd --stdin ftptest
```

接下来当然就是安装 vsftpd 这只服务器软件，同时启动这只服务，另外，我们也希望未来开机都能够启动这只服务！ 因此需要这样做 （鸟哥假设你的 CentOS 7.x 的原版光盘已经挂载于 /mnt 了喔！）：

```
[root@study ~]# yum install /mnt/Packages/vsftpd-3*
[root@study ~]# systemctl start vsftpd
[root@study ~]# systemctl enable vsftpd
[root@study ~]# netstat -tlnp
Active Internet connections （only servers）
Proto Recv-Q Send-Q Local Address   Foreign Address   State   PID/Program name
tcp        0      0 0.0.0.0:22      0.0.0.0:*         LISTEN  1326/sshd
tcp        0      0 127.0.0.1:25    0.0.0.0:*         LISTEN  2349/master
tcp6       0      0 :::21           :::*              LISTEN  6256/vsftpd
tcp6       0      0 :::22           :::*              LISTEN  1326/sshd
tcp6       0      0 ::1:25          :::*              LISTEN  2349/master
# 要注意看，上面的特殊字体那行有出现，才代表 vsftpd 这只服务有启动喔！！
```

- 匿名者无法下载的问题

现在让我们来仿真一些 FTP 的常用状态！假设你想要将 /etc/securetty 以及主要的 /etc/sysctl.conf 放置给所有人下载， 那么你可能会这样做！

```
[root@study ~]# cp -a /etc/securetty /etc/sysctl.conf /var/ftp/pub
[root@study ~]# ll /var/ftp/pub
-rw-------. 1 root root 221 Oct 29  2014 securetty    # 先假设你没有看到这个问题！
-rw-r--r--. 1 root root 225 Mar  6 11:05 sysctl.conf
```

一般来说，默认要给用户下载的 FTP 文件会放置到上面表格当中的 /var/ftp/pub 目录喔！现在让我们使用简单的终端机浏览器 curl 来观察看看！ 看你能不能查询到上述两个文件的内容呢？

```
# 1\. 先看看 FTP 根目录下面有什么文件存在？
[root@study ~]# curl ftp://localhost
drwxr-xr-x    2 0        0              40 Aug 08 00:51 pub
# 确实有存在一个名为 pub 的文件喔！那就是在 /var/ftp 下面的 pub 啰！

# 2\. 再往下看看，能不能看到 pub 内的文件呢？
[root@study ~]# curl ftp://localhost/pub/  # 因为是目录，要加上 / 才好！
-rw-------    1 0        0             221 Oct 29  2014 securetty
-rw-r--r--    1 0        0             225 Mar 06 03:05 sysctl.conf

# 3\. 承上，继续看一下 sysctl.conf 的内容好了！
[root@study ~]# curl ftp://localhost/pub/sysctl.conf
# System default settings live in /usr/lib/sysctl.d/00-system.conf.
# To override those settings, enter new settings here, or in an /etc/sysctl.d/&lt;name&gt
#
# For more information, see sysctl.conf（5） and sysctl.d（5）.
# 眞的有看到这个文件的内容喔！所以确定是可以让 vsftpd 读取到这文件的！

# 4\. 再来瞧瞧 securetty 好了！
[root@study ~]# curl ftp://localhost/pub/securetty
curl: （78） RETR response: 550
# 看不到耶！但是，基本的原因应该是权限问题喔！因为 vsftpd 默认放在 /var/ftp/pub 内的数据，
# 不论什么 SELinux type 几乎都可以被读取的才对喔！所以要这样处理！

# 5\. 修订权限之后再一次观察 securetty 看看！
[root@study ~]# chmod a+r /var/ftp/pub/securetty
[root@study ~]# curl ftp://localhost/pub/securetty
# 此时你就可以看到实际的文件内容啰！

# 6\. 修订 SELinux type 的内容 （非必备）
[root@study ~]# restorecon -Rv /var/ftp
```

上面这个例子在告诉你，要先从权限的角度来瞧一瞧，如果无法被读取，可能就是因为没有 r 或没有 rx 啰！并不一定是由 SELinux 引起的！ 了解乎？好～再来瞧瞧如果是一般帐号呢？如何登陆？

- 无法从主文件夹下载文件的问题分析与解决

我们前面创建了 ftptest 帐号，那如何使用文字界面来登陆呢？就使用如下的方式来处理。同时请注意，因为文字体的 FTP 用户端软件， 默认会将用户丢到根目录而不是主文件夹，因此，你的 URL 可能需要修订一下如下！

```
# 0\. 为了让 curl 这个文字浏览器可以传输数据，我们先创建一些数据在 ftptest 主文件夹
[root@study ~]# echo "testing" &gt; ~ftptest/test.txt
[root@study ~]# cp -a /etc/hosts /etc/sysctl.conf ~ftptest/
[root@study ~]# ll ~ftptest/
-rw-r--r--. 1 root root 158 Jun  7  2013 hosts
-rw-r--r--. 1 root root 225 Mar  6 11:05 sysctl.conf
-rw-r--r--. 1 root root   8 Aug  9 01:05 test.txt
```

```
# 1\. 一般帐号直接登陆 FTP 服务器，同时变换目录到主文件夹去！
[root@study ~]# curl ftp://ftptest:myftp123@localhost/~/
-rw-r--r--    1 0        0             158 Jun 07  2013 hosts
-rw-r--r--    1 0        0             225 Mar 06 03:05 sysctl.conf
-rw-r--r--    1 0        0               8 Aug 08 17:05 test.txt
# 真的有数据～看文件最左边的权限也是没问题，所以，来读一下 test.txt 的内容看看

# 2\. 开始下载 test.txt, sysctl.conf 等有权限可以阅读的文件看看！
[root@study ~]# curl ftp://ftptest:myftp123@localhost/~/test.txt
curl: (78) RETR response: 550
# 竟然说没有权限！明明我们的 rwx 是正常没问题！那是否有可能是 SELinux 造成的？

# 3\. 先将 SELinux 从 Enforce 转成 Permissive 看看情况！同时观察登录文件
[root@study ~]# setenforce 0
[root@study ~]# curl ftp://ftptest:myftp123@localhost/~/test.txt
testing
[root@study ~]# setenforce 1  # 确定问题后，一定要转成 Enforcing 啊！
# 确定有数据内容！所以，确定就是 SELinux 造成无法读取的问题～那怎办？要改规则？还是改 type？
# 因为都不知道，所以，就检查一下登录文件看看有没有相关的信息可以提供给我们处理！

[root@study ~]# vim /var/log/messages
Aug  9 02:55:58 station3-39 setroubleshoot: SELinux is preventing /usr/sbin/vsftpd
 from lock access on the file /home/ftptest/test.txt. For complete SELinux messages.
 run sealert -l 3a57aad3-a128-461b-966a-5bb2b0ffa0f9
Aug  9 02:55:58 station3-39 python: SELinux is preventing /usr/sbin/vsftpd from
 lock access on the file /home/ftptest/test.txt.

*****  Plugin catchall_boolean (47.5 confidence) suggests   ******************

If you want to allow ftp to home dir
Then you must tell SELinux about this by enabling the 'ftp_home_dir' boolean.
You can read 'None' man page for more details.
Do
setsebool -P ftp_home_dir 1

*****  Plugin catchall_boolean (47.5 confidence) suggests   ******************

If you want to allow ftpd to full access
Then you must tell SELinux about this by enabling the 'ftpd_full_access' boolean.
You can read 'None' man page for more details.
Do
setsebool -P ftpd_full_access 1

*****  Plugin catchall (6.38 confidence) suggests   **************************
.....(下面省略).....
# 基本上，你会看到有个特殊字体的部份，就是 sealert 那一行。虽然下面已经列出可能的解决方案了，
# 就是一堆底线那些东西。至少就有三个解决方案 (最后一个没列出来)，哪种才是正确的？
# 为了了解正确的解决方案，我们还是还执行一下 sealert 那行吧！看看情况再说！

# 4\. 通过 sealert 的解决方案来处理问题
[root@study ~]# sealert -l 3a57aad3-a128-461b-966a-5bb2b0ffa0f9
SELinux is preventing /usr/sbin/vsftpd from lock access on the file /home/ftptest/test.tx

# 下面说有 47.5% 的概率是由于这个原因所发生，并且可以使用 setsebool 去解决的意思！
*****  Plugin catchall_boolean (47.5 confidence) suggests   ******************

If you want to allow ftp to home dir
Then you must tell SELinux about this by enabling the 'ftp_home_dir' boolean.
You can read 'None' man page for more details.
Do
setsebool -P ftp_home_dir 1

# 下面说也是有 47.5% 的概率是由此产生的！
*****  Plugin catchall_boolean (47.5 confidence) suggests   ******************

If you want to allow ftpd to full access
Then you must tell SELinux about this by enabling the 'ftpd_full_access' boolean.
You can read 'None' man page for more details.
Do
setsebool -P ftpd_full_access 1
```

```
# 下面说，仅有 6.38% 的可信度是由这个情况产生的！
*****  Plugin catchall （6.38 confidence） suggests   **************************

If you believe that vsftpd should be allowed lock access on the test.txt file by default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:
# grep vsftpd /var/log/audit/audit.log &#124; audit2allow -M mypol
# semodule -i mypol.pp

# 下面就重要了！是整个问题发生的主因～最好还是稍微瞧一瞧！
Additional Information:
Source Context                system_u:system_r:ftpd_t:s0-s0:c0.c1023
Target Context                unconfined_u:object_r:user_home_t:s0
Target Objects                /home/ftptest/test.txt [ file ]
Source                        vsftpd
Source Path                   /usr/sbin/vsftpd
Port                          &lt;Unknown&gt;
Host                          station3-39.gocloud.vm
Source RPM Packages           vsftpd-3.0.2-9.el7.x86_64
Target RPM Packages
Policy RPM                    selinux-policy-3.13.1-23.el7.noarch
Selinux Enabled               True
Policy Type                   targeted
Enforcing Mode                Permissive
Host Name                     station3-39.gocloud.vm
Platform                      Linux station3-39.gocloud.vm 3.10.0-229.el7.x86_64
                              #1 SMP Fri Mar 6 11:36:42 UTC 2015 x86_64 x86_64
Alert Count                   3
First Seen                    2015-08-09 01:00:12 CST
Last Seen                     2015-08-09 02:55:57 CST
Local ID                      3a57aad3-a128-461b-966a-5bb2b0ffa0f9

Raw Audit Messages
type=AVC msg=audit（1439060157.358:635）: avc:  denied  { lock } for  pid=5029 comm="vsftp
 path="/home/ftptest/test.txt" dev="dm-2" ino=141 scontext=system_u:system_r:ftpd_t:s0-s0
 c0.c1023 tcontext=unconfined_u:object_r:user_home_t:s0 tclass=file

type=SYSCALL msg=audit（1439060157.358:635）: arch=x86_64 syscall=fcntl success=yes exit=0
 a0=4 a1=7 a2=7fffceb8cbb0 a3=0 items=0 ppid=5024 pid=5029 auid=4294967295 uid=1001 gid=1
 euid=1001 suid=1001 fsuid=1001 egid=1001 sgid=1001 fsgid=1001 tty=（none） ses=4294967295
 comm=vsftpd exe=/usr/sbin/vsftpd subj=system_u:system_r:ftpd_t:s0-s0:c0.c1023 key=（null

Hash: vsftpd,ftpd_t,user_home_t,file,lock
```

经过上面的测试，现在我们知道主要的问题发生在 SELinux 的 type 不是 vsftpd_t 所能读取的原因～ 经过仔细观察 test.txt 文件的类型，我们知道他原本就是主文件夹，因此是 user_home_t 也没啥了不起的啊！是正确的～ 因此，分析两个比较可信 （47.5%） 的解决方案后，可能是与 ftp_home_dir 比较有关啊！所以，我们应该不需要修改 SELinux type， 修改的应该是 SELinux rules 才对！所以，这样做看看：

```
# 1\. 先确认一下 SELinux 的模式，然后再瞧一瞧能否下载 test.txt，最终使用处理方式来解决～
[root@study ~]# getenforce
Enforcing
[root@study ~]# curl ftp://ftptest:myftp123@localhost/~/test.txt
curl: （78） RETR response: 550
# 确定还是无法读取的喔！
[root@study ~]# setsebool -P ftp_home_dir 1
[root@study ~]# curl ftp://ftptest:myftp123@localhost/~/test.txt
testing
# OK！太赞了！处理完毕！现在使用者可以在自己的主文件夹上传/下载文件了！

# 2\. 开始下载其他文件试看看啰！
[root@study ~]# curl ftp://ftptest:myftp123@localhost/~/sysctl.conf
# System default settings live in /usr/lib/sysctl.d/00-system.conf.
# To override those settings, enter new settings here, or in an /etc/sysctl.d/&lt;name&gt
#
# For more information, see sysctl.conf（5） and sysctl.d（5）.
```

没问题喔！通过修改 SELinux rule 的布林值，现在我们就可以使用一般帐号在 FTP 服务来上传/下载数据啰！非常愉快吧！ 那万一我们还有其他的目录也想要通过 FTP 来提供这个 ftptest 用户上传与下载呢？往下瞧瞧～

- 一般帐号用户从非正规目录上传/下载文件

假设我们还想要提供 /srv/gogogo 这个目录给 ftptest 用户使用，那又该如何处理呢？假设我们都没有考虑 SELinux ， 那就是这样的情况：

```
# 1\. 先处理好所需要的目录数据
[root@study ~]# mkdir /srv/gogogo
[root@study ~]# chgrp ftptest /srv/gogogo
[root@study ~]# echo "test" &gt; /srv/gogogo/test.txt

# 2\. 开始直接使用 ftp 观察一下数据！
[root@study ~]# curl ftp://ftptest:myftp123@localhost//srv/gogogo/test.txt
curl: （78） RETR response: 550
# 有问题喔！来瞧瞧登录文件怎么说！
[root@study ~]# grep sealert /var/log/messages &#124; tail
Aug  9 04:23:12 station3-39 setroubleshoot: SELinux is preventing /usr/sbin/vsftpd from
 read access on the file test.txt. For complete SELinux messages. run sealert -l
 08d3c0a2-5160-49ab-b199-47a51a5fc8dd
[root@study ~]# sealert -l 08d3c0a2-5160-49ab-b199-47a51a5fc8dd
SELinux is preventing /usr/sbin/vsftpd from read access on the file test.txt.

# 虽然这个可信度比较高～不过，因为会全部放行 FTP ，所以不太考虑！
*****  Plugin catchall_boolean （57.6 confidence） suggests   ******************

If you want to allow ftpd to full access
Then you must tell SELinux about this by enabling the 'ftpd_full_access' boolean.
You can read 'None' man page for more details.
Do
setsebool -P ftpd_full_access 1

# 因为是非正规目录的使用，所以这边加上默认 SELinux type 恐怕会是比较正确的选择！
*****  Plugin catchall_labels （36.2 confidence） suggests   *******************

If you want to allow vsftpd to have read access on the test.txt file
Then you need to change the label on test.txt
Do
# semanage fcontext -a -t FILE_TYPE 'test.txt'
where FILE_TYPE is one of the following: NetworkManager_tmp_t, abrt_helper_exec_t, abrt_t
 abrt_upload_watch_tmp_t, abrt_var_cache_t, abrt_var_run_t, admin_crontab_tmp_t, afs_cach
 alsa_home_t, alsa_tmp_t, amanda_tmp_t, antivirus_home_t, antivirus_tmp_t, apcupsd_tmp_t,
Then execute:
restorecon -v 'test.txt'

*****  Plugin catchall （7.64 confidence） suggests   **************************

If you believe that vsftpd should be allowed read access on the test.txt file by default.
Then you should report this as a bug.
You can generate a local policy module to allow this access.
Do
allow this access for now by executing:
# grep vsftpd /var/log/audit/audit.log &#124; audit2allow -M mypol
# semodule -i mypol.pp

Additional Information:
Source Context                system_u:system_r:ftpd_t:s0-s0:c0.c1023
Target Context                unconfined_u:object_r:var_t:s0
Target Objects                test.txt [ file ]
Source                        vsftpd
.....（下面省略）.....
```

因为是非正规目录啊，所以感觉上似乎与 semanage 那一行的解决方案比较相关～接下来就是要找到 FTP 的 SELinux type 来解决啰！ 所以，让我们查一下 FTP 相关的数据啰！

```
# 3\. 先查看一下 /var/ftp 这个地方的 SELinux type 吧！
[root@study ~]# ll -Zd /var/ftp
drwxr-xr-x. root root system_u:object_r:public_content_t:s0 /var/ftp

# 4\. 以 sealert 建议的方法来处理好 SELinux type 啰！
[root@study ~]# semanage fcontext -a -t public_content_t "/srv/gogogo（/.*）?"
[root@study ~]# restorecon -Rv /srv/gogogo
[root@study ~]# curl ftp://ftptest:myftp123@localhost//srv/gogogo/test.txt
test
# 喔耶！终于再次搞定喔！
```

在这个范例中，我们是修改了 SELinux type 喔！与前一个修改 SELinux rule 不太一样！要理解理解喔！

- 无法变更 FTP 连线端口问题分析与解决

在某些情况下，可能你的服务器软件需要开放在非正规的端口，举例来说，如果因为某些政策问题，导致 FTP 启动的正常的 21 号端口无法使用， 因此你想要启用在 555 号端口时，该如何处理呢？基本上，既然 SELinux 的主体程序大多是被受限的网络服务，没道理不限制放行的端口啊！ 所以，很可能会出问题～那就得要想想办法才行！

```
# 1\. 先处理 vsftpd 的配置文件，加入换 port 的参数才行！
[root@study ~]# vim /etc/vsftpd/vsftpd.conf
# 请按下大写的 G 跑到最后一行，然后新增加下面这行设置！前面不可以留白！
listen_port=555

# 2\. 重新启动 vsftpd 并且观察登录文件的变化！
[root@study ~]# systemctl restart vsftpd
[root@study ~]# grep sealert /var/log/messages
Aug  9 06:34:46 station3-39 setroubleshoot: SELinux is preventing /usr/sbin/vsftpd from
 name_bind access on the tcp_socket port 555\. For complete SELinux messages. run
 sealert -l 288118e7-c386-4086-9fed-2fe78865c704

[root@study ~]# sealert -l 288118e7-c386-4086-9fed-2fe78865c704
SELinux is preventing /usr/sbin/vsftpd from name_bind access on the tcp_socket port 555.

*****  Plugin bind_ports （92.2 confidence） suggests   ************************

If you want to allow /usr/sbin/vsftpd to bind to network port 555
Then you need to modify the port type.
Do
# semanage port -a -t PORT_TYPE -p tcp 555
    where PORT_TYPE is one of the following: certmaster_port_t, cluster_port_t,
 ephemeral_port_t, ftp_data_port_t, ftp_port_t, hadoop_datanode_port_t, hplip_port_t,
 port_t, postgrey_port_t, unreserved_port_t.
.....（后面省略）.....
# 看一下信任度，高达 92.2% 耶！几乎就是这家伙～因此不必再看～就是他了！比较重要的是，
# 解决方案里面，那个 PORT_TYPE 有很多选择～但我们是要打开 FTP 端口嘛！所以，
# 就由后续数据找到 ftp_port_t 那个项目啰！带入实验看看！

# 3\. 实际带入 SELinux 端口修订后，在重新启动 vsftpd 看看
[root@study ~]# semanage port -a -t ftp_port_t -p tcp 555
[root@study ~]# systemctl restart vsftpd
[root@study ~]# netstat -tlnp
Active Internet connections （only servers）
Proto Recv-Q Send-Q Local Address    Foreign Address   State    PID/Program name
tcp        0      0 0.0.0.0:22       0.0.0.0:*         LISTEN   1167/sshd
tcp        0      0 127.0.0.1:25     0.0.0.0:*         LISTEN   1598/master
tcp6       0      0 :::555           :::*              LISTEN   8436/vsftpd
tcp6       0      0 :::22            :::*              LISTEN   1167/sshd
tcp6       0      0 ::1:25           :::*              LISTEN   1598/master

# 4\. 实验看看这个 port 能不能用？
[root@study ~]# curl ftp://localhost:555/pub/
-rw-r--r--    1 0        0             221 Oct 29  2014 securetty
-rw-r--r--    1 0        0             225 Mar 06 03:05 sysctl.conf
```

通过上面的几个小练习，你会知道在正规或非正规的环境下，如何处理你的 SELinux 问题哩！仔细研究看看啰！

16.6 重点回顾

- 程序 （program）：通常为 binary program ，放置在储存媒体中 （如硬盘、光盘、软盘、磁带等），为实体文件的型态存在；
- 程序 （process）：程序被触发后，执行者的权限与属性、程序的程序码与所需数据等都会被载入内存中， 操作系统并给予这个内存内的单元一个识别码 （PID），可以说，程序就是一个正在运行中的程序。
- 程序彼此之间是有相关性的，故有父程序与子程序之分。而 Linux 系统所有程序的父程序就是 init 这个 PID 为 1 号的程序。
- 在 Linux 的程序调用通常称为 fork-and-exec 的流程！程序都会借由父程序以复制 （fork） 的方式产生一个一模一样的子程序， 然后被复制出来的子程序再以 exec 的方式来执行实际要进行的程序，最终就成为一个子程序的存在。
- 常驻在内存当中的程序通常都是负责一些系统所提供的功能以服务使用者各项任务，因此这些常驻程序就会被我们称为：服务 （daemon）。
- 在工作管理 （job control） 中，可以出现提示字符让你操作的环境就称为前景 （foreground），至于其他工作就可以让你放入背景 （background） 去暂停或运行。
- 与 job control 有关的按键与关键字有： &, [ctrl]-z, jobs, fg, bg, kill %n 等；
- 程序管理的观察指令有： ps, top, pstree 等等；
- 程序之间是可以互相控制的，传递的讯息 （signal） 主要通过 kill 这个指令在处理；
- 程序是有优先顺序的，该项目为 Priority，但 PRI 是核心动态调整的，使用者只能使用 nice 值去微调 PRI
- nice 的给予可以有： nice, renice, top 等指令；
- vmstat 为相当好用的系统资源使用情况观察指令；
- SELinux 当初的设计是为了避免使用者资源的误用，而 SELinux 使用的是 MAC 委任式存取设置；
- SELinux 的运行中，重点在于主体程序 （Subject） 能否存取目标文件资源 （Object），这中间牵涉到政策 （Policy） 内的规则， 以及实际的安全性本文类别 （type）；
- 安全性本文的一般设置为：“Identify:role:type”其中又以 type 最重要；
- SELinux 的模式有： enforcing, permissive, disabled 三种，而启动的政策 （Policy） 主要是 targeted
- SELinux 启动与关闭的配置文件在： /etc/selinux/config
- SELinux 的启动与观察： getenforce, sestatus 等指令
- 重设 SELinux 的安全性本文可使用 restorecon 与 chcon
- 在 SELinux 有启动时，必备的服务至少要启动 auditd 这个！
- 若要管理默认的 SELinux 布林值，可使用 getsebool, setsebool 来管理！

16.7 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ）

- 简单说明什么是程序 （program） 而什么是程序 （process）？程序 （program） 是系统上面可以被执行的文件，由于 Linux 的完整文件名 （由 / 写起） 仅能有一个， 所以 program 的文件名具有单一性。当程序被执行后，就会启动成程序 （process）， 一个 program 可以被不同的使用者或者相同的使用者重复的执行成为多个程序， 且该程序所造成的程序还因为不同的使用者，而有不同的权限，且每个 process 几乎都是独立的。
- 我今天想要查询 /etc/crontab 与 crontab 这个程序的用法与写法，请问我该如何线上查询？查询 crontab 指令可以使用 man crontab 或 info crontab ，至于查询 /etc/crontab ，则可以使用 man 5 crontab 啰！
- 我要如何查询 crond 这个 daemon 的 PID 与他的 PRI 值呢？ps -lA | grep crond 即可查到！
- 我要如何修改 crond 这个 PID 的优先执行序？先以 ps aux 找到 crond 的 PID 后，再以： renice -n number PID 来调整！
- 我是一般身份使用者，我是否可以调整不属于我的程序的 nice 值？此外，如果我调整了我自己的程序的 nice 值到 10 ，是否可以将他调回 5 呢？不行！一般身份使用者仅能调整属于自己的 PID 程序，并且，只能将 nice 值一再地调高，并不能调低，所以调整为 10 之后，就不能降回 5 啰！
- 我要怎么知道我的网卡在开机的过程中有没有被捉到？可以使用 dmesg 来视察！

16.8 参考资料与延伸阅读

- [1]关于 fork-and-exec 的说明可以参考如下网页与书籍： 吴贤明老师维护的网站：http://nmc.nchu.edu.tw/linux/process.htm 杨振和、操作系统导论、第三章、学贯出版社
- [2]对 Linux 核心有兴趣的话，可以先看看下面的链接：http://www.linux.org.tw/CLDP/OLD/INFO-SHEET-2.html
- [3]来自 Linux Journal 的关于 /proc 的说明：http://www.linuxjournal.com/article/177
- [4]关于 SELinux 相关的网站与文件数据： 美国国家安全局的 SELinux 简介：http://www.nsa.gov/research/selinux/ 陈永升、“企业级Linux 系统管理宝典”、学贯行销股份有限公司 Fedora SELinux 说明：http://fedoraproject.org/wiki/SELinux/SecurityContext 美国国家安全局对 SELinux 的白皮书：http://www.nsa.gov/research/_files/selinux/papers/module/t1.shtml

2002/06/28：第一次完成 2003/02/10：重新编排与加入 FAQ 2005/09/07：将旧的文章移动到此处 。 2005/09/18：哈哈，终于将这篇写完啰。新增了一些简单的小指令啦。 2009/03/15：将旧的基于 FC4 的文章移动到此处。 2009/03/19：调整 sar 成为 vmstat ，因为 vmstat 是默认有安装的分析工具！ 2009/09/11：加入了 nohup 的说明啰！并加入了情境仿真题 2011/04/14：原本的习题解答为 ps aux ，应该是 ps -lA 才好！感谢网友 redsc 的回报！ 2012/06/14：原本是“内存字段（procs）”错了！是“程序字段”才对！ 2013/08/02：在 signal 的相关说明中，SIGSTOP 之前写错了！应该是 19 号才对！请观察 man 7 signal（感谢网友王兄来信说明） 2015/08/03：将旧的基于 CentOS 5.x 的版本移动到 此处，有需要请前往查阅。

第十七章、认识系统服务 （daemons）

最近更新日期：20//

在 Unix-Like 的系统中，你会常常听到 daemon 这个字眼！那么什么是传说中的 daemon 呢？这些 daemon 放在什么地方？他的功能是什么？该如何启动这些 daemon ？又如何有效的将这些 daemon 管理妥当？此外，要如何视察这些 daemon 开了多少个 ports ？又这些 ports 要如何关闭？还有还有，晓得你系统的这些 port 各代表的是什么服务吗？ 这些都是最基础需要注意的呢！尤其是在架设网站之前，这里的观念就显的更重要了。

从 CentOS 7.x 这一版之后，传统的 init 已经被舍弃，取而代之的是 systemd 这个家伙～这家伙跟之前的 init 有什么差异？ 优缺点为何？如何管理不同种类的服务类型？以及如何取代原本的“执行等级”等等，很重要的改变喔！

17.1 什么是 daemon 与服务 （service）

我们在第十六章就曾经谈过“服务”这东西！ 当时的说明是“常驻在记体体中的程序，且可以提供一些系统或网络功能，那就是服务”。而服务一般的英文说法是“ service ”。

但如果你常常上网去查看一些数据的话，尤其是 Unix-Like 的相关操作系统，应该常常看到“请启动某某 daemon 来提供某某功能”，唔！那么 daemon 与 service 有关啰？否则为什么都能够提供某些系统或网络功能？此外，这个 daemon 是什么东西呀？ daemon 的字面上的意思就是“守护神、恶魔？”还真是有点奇怪呦！^_^""！

简单的说，系统为了某些功能必须要提供一些服务 （不论是系统本身还是网络方面） ，这个服务就称为 service 。 但是 service 的提供总是需要程序的运行吧！否则如何执行呢？所以达成这个 service 的程序我们就称呼他为 daemon 啰！ 举例来说，达成循环型例行性工作调度服务 （service） 的程序为 crond 这个 daemon 啦！这样说比较容易理解了吧！



Tips 你不必去区分什么是 daemon 与 service ！事实上，你可以将这两者视为相同！因为达成某个服务是需要一支 daemon 在背景中运行， 没有这支 daemon 就不会有 service ！所以不需要分的太清楚啦！

一般来说，当我们以文字模式或图形模式 （非单人维护模式） 完整开机进入 Linux 主机后，系统已经提供我们很多的服务了！包括打印服务、工作调度服务、邮件管理服务等等； 那么这些服务是如何被启动的？他们的工作型态如何？下面我们就来谈一谈啰！



Tips daemon 既然是一只程序执行后的程序，那么 daemon 所处的那个原本的程序通常是如何命名的呢 （daemon 程序的命名方式） 。 每一个服务的开发者，当初在开发他们的服务时，都有特别的故事啦！不过，无论如何，这些服务的名称被创建之后，被挂上 Linux 使用时，通常在服务的名称之后会加上一个 d ，例如例行性命令的创建的 at, 与 cron 这两个服务， 他的程序文件名会被取为 atd 与 crond，这个 d 代表的就是 daemon 的意思。所以，在第十六章中，我们使用了 ps 与 top 来观察程序时，都会发现到很多的 {xxx}d 的程序，呵呵！通常那就是一些 daemon 的程序啰！

17.1.1 早期 System V 的 init 管理行为中 daemon 的主要分类（Optional）

还记得我们在第一章谈到过 Unix 的 system V 版本吧？那个很纯种的 Unix 版本～ 在那种年代下面，我们启动系统服务的管理方式被称为 SysV 的 init 脚本程序的处理方式！亦即系统核心第一支调用的程序是 init ， 然后 init 去唤起所有的系统所需要的服务，不论是本机服务还是网络服务就是了。

基本上 init 的管理机制有几个特色如下：

- 服务的启动、关闭与观察等方式： 所有的服务启动脚本通通放置于 /etc/init.d/ 下面，基本上都是使用 bash shell script 所写成的脚本程序，需要启动、关闭、重新启动、观察状态时， 可以通过如下的方式来处理：
  - 启动：/etc/init.d/daemon start
  - 关闭：/etc/init.d/daemon stop
  - 重新启动：/etc/init.d/daemon restart
  - 状态观察：/etc/init.d/daemon status
- 服务启动的分类： init 服务的分类中，依据服务是独立启动或被一只总管程序管理而分为两大类：
  - 独立启动模式 （stand alone）：服务独立启动，该服务直接常驻于内存中，提供本机或用户的服务行为，反应速度快。
  - 总管程序 （super daemon）：由特殊的 xinetd 或 inetd 这两个总管程序提供 socket 对应或 port 对应的管理。当没有用户要求某 socket 或 port 时， 所需要的服务是不会被启动的。若有用户要求时， xinetd 总管才会去唤醒相对应的服务程序。当该要求结束时，这个服务也会被结束掉～ 因为通过 xinetd 所总管，因此这个家伙就被称为 super daemon。好处是可以通过 super daemon 来进行服务的时程、连线需求等的控制，缺点是唤醒服务需要一点时间的延迟。
- 服务的相依性问题： 服务是可能会有相依性的～例如，你要启动网络服务，但是系统没有网络， 那怎么可能可以唤醒网络服务呢？如果你需要连线到外部取得认证服务器的连线，但该连线需要另一个A服务的需求，问题是，A服务没有启动， 因此，你的认证服务就不可能会成功启动的！这就是所谓的服务相依性问题。init 在管理员自己手动处理这些服务时，是没有办法协助相依服务的唤醒的！
- 执行等级的分类： 上面说到 init 是开机后核心主动调用的， 然后 init 可以根据使用者自订的执行等级 （runlevel） 来唤醒不同的服务，以进入不同的操作界面。基本上 Linux 提供 7 个执行等级，分别是 0, 1, 2...6 ， 比较重要的是 1）单人维护模式、3）纯文本模式、5）文字加图形界面。而各个执行等级的启动脚本是通过 /etc/rc.d/rc[0-6]/SXXdaemon 链接到 /etc/init.d/daemon ， 链接文件名 （SXXdaemon） 的功能为： S 为启动该服务，XX是数字，为启动的顺序。由于有 SXX 的设置，因此在开机时可以“依序执行”所有需要的服务， 同时也能解决相依服务的问题。这点与管理员自己手动处理不太一样就是了。
- 制定执行等级默认要启动的服务： 若要创建如上提到的 SXXdaemon 的话，不需要管理

员手动创建链接文件，通过如下的指令可以来处理默认启动、默认不启动、观察默认启动否的行为：
  - 默认要启动： chkconfig daemon on
  - 默认不启动： chkconfig daemon off
  - 观察默认为启动否： chkconfig --list daemon
- 执行等级的切换行为： 当你要从纯命令行 （runlevel 3） 切换到图形界面 （runlevel 5）， 不需要手动启动、关闭该执行等级的相关服务，只要“ init 5 ”即可切换，init 这小子会主动去分析 /etc/rc.d/rc[35].d/ 这两个目录内的脚本， 然后启动转换 runlevel 中需要的服务～就完成整体的 runlevel 切换。

基本上 init 主要的功能都写在上头了，重要的指令包括 daemon 本身自己的脚本 （/etc/init.d/daemon） 、xinetd 这个特殊的总管程序 （super daemon） 、设置默认开机启动的 chkconfig， 以及会影响到执行等级的 init N 等。虽然 CentOS 7 已经不使用 init 来管理服务了，不过因为考虑到某些脚本没有办法直接塞入 systemd 的处理，因此这些脚本还是被保留下来， 所以，我们在这里还是稍微介绍了一下。更多更详细的数据就请自己查询旧版本啰！如下就是一个可以参考的版本：

- http://linux.vbird.org/linux_basic/0560daemons/0560daemons-centos5.php

17.1.2 systemd 使用的 unit 分类

从 CentOS 7.x 以后，Red Hat 系列的 distribution 放弃沿用多年的 System V 开机启动服务的流程，就是前一小节提到的 init 启动脚本的方法， 改用 systemd 这个启动服务管理机制～那么 systemd 有什么好处呢？

- 平行处理所有服务，加速开机流程： 旧的 init 启动脚本是“一项一项任务依序启动”的模式，因此不相依的服务也是得要一个一个的等待。但目前我们的硬件主机系统与操作系统几乎都支持多核心架构了， 没道理未相依的服务不能同时启动啊！systemd 就是可以让所有的服务同时启动，因此你会发现到，系统启动的速度变快了！
- 一经要求就回应的 on-demand 启动方式： systemd 全部就是仅有一只 systemd 服务搭配 systemctl 指令来处理，无须其他额外的指令来支持。不像 systemV 还要 init, chkconfig, service... 等等指令。 此外， systemd 由于常驻内存，因此任何要求 （on-demand） 都可以立即处理后续的 daemon 启动的任务。
- 服务相依性的自我检查： 由于 systemd 可以自订服务相依性的检查，因此如果 B 服务是架构在 A 服务上面启动的，那当你在没有启动 A 服务的情况下仅手动启动 B 服务时， systemd 会自动帮你启动 A 服务喔！这样就可以免去管理员得要一项一项服务去分析的麻烦～（如果读者不是新手，应该会有印象，当你没有启动网络， 但却启动 NIS/NFS 时，那个开机时的 timeout 甚至可达到 10~30 分钟...）
- 依 daemon 功能分类： systemd 旗下管理的服务非常多，包山包海啦～为了厘清所有服务的功能，因此，首先 systemd 先定义所有的服务为一个服务单位 （unit），并将该 unit 归类到不同的服务类型 （type） 去。 旧的 init 仅分为 stand alone 与 super daemon 实在不够看，systemd 将服务单位 （unit） 区分为 service, socket, target, path,

snapshot, timer 等多种不同的类型（type），方便管理员的分类与记忆。

- 将多个 daemons 集合成为一个群组：如同 systemV 的 init 里头有个 runlevel 的特色，systemd 亦将许多的功能集合成为一个所谓的 target 项目，这个项目主要在设计操作环境的创建，所以是集合了许多的 daemons，亦即是执行某个 target 就是执行好多个 daemon 的意思！
- 向下相容旧有的 init 服务脚本：基本上，systemd 是可以相容于 init 的启动脚本的，因此，旧的 init 启动脚本也能够通过 systemd 来管理，只是更进阶的 systemd 功能就没有办法支持就是了。

虽然如此，不过 systemd 也是有些地方无法完全取代 init 的！包括：

- 在 runlevel 的对应上，大概仅有 runlevel 1, 3, 5 有对应到 systemd 的某些 target 类型而已，没有全部对应；
- 全部的 systemd 都用 systemctl 这个管理程序管理，而 systemctl 支持的语法有限制，不像 /etc/init.d/daemon 就是纯脚本可以自订参数，systemctl 不可自订参数。；
- 如果某个服务启动是管理员自己手动执行启动，而不是使用 systemctl 去启动的（例如你自己手动输入 crond 以启动 crond 服务），那么 systemd 将无法侦测到该服务，而无法进一步管理。
- systemd 启动过程中，无法与管理员通过 standard input 传入讯息！因此，自行撰写 systemd 的启动设置时，务必要取消互动机制～（连通过启动时传进的标准输入讯息也要避免！）

不过，光是同步启动服务脚本这个功能就可以节省你很多开机的时间～同时 systemd 还有很多特殊的服务类型（type）可以提供更多有趣的功能！确实值得学一学～ 而且 CentOS 7 已经用了 systemd 了！想不学也不行啊～哈哈哈！好～既然要学，首先就得要针对 systemd 管理的 unit 来了解一下。

- systemd 的配置文件放置目录

基本上，systemd 将过去所谓的 daemon 执行脚本通通称为一个服务单位（unit），而每种服务单位依据功能来区分时，就分类为不同的类型（type）。基本的类型有包括系统服务、数据监听与交换的插槽档服务（socket）、储存系统状态的快照类型、提供不同类似执行等级分类的操作环境（target）等等。哇！这么多类型，那设置时会不会很麻烦呢？其实还好，因为配置文件都放置在下面的目录中：

- /usr/lib/systemd/system/：每个服务最主要的启动脚本设置，有点类似以前的 /etc/init.d 下面的文件；
- /run/systemd/system/：系统执行过程中所产生的服务脚本，这些脚本的优先序要比 /usr/lib/systemd/system/ 高！
- /etc/systemd/system/：管理员依据主机系统的需求所创建的执行脚本，其实这个目录有点像以前 /etc/rc.d/rc5.d/Sxx 之类的功能！执行优先序又比 /run/systemd/system/ 高喔！

也就是说，到底系统开机会不会执行某些服务其实是看 /etc/systemd/system/ 下面的设置，所以该目录下面就是一大堆链接文件。而实际执行的 systemd 启动脚本配置文件， 其实都是放置在 /usr/lib/systemd/system/ 下面的喔！因此如果你想要修改某个服务启动的设置，应该要去 /usr/lib/systemd/system/ 下面修改才对！ /etc/systemd/system/ 仅是链接到正确的执行脚本配置文件而已。所以想要看执行脚本设置，应该就得要到 /usr/lib/systemd/system/ 下面去查阅才对！

- systemd 的 unit 类型分类说明

那 /usr/lib/systemd/system/ 以下的数据如何区分上述所谓的不同的类型 （type） 呢？很简单！看扩展名！举例来说，我们来瞧瞧上一章谈到的 vsftpd 这个范例的启动脚本设置， 还有 crond 与纯文本模式的 multi-user 设置：

```
[root@study ~]# ll /usr/lib/systemd/system/ &#124; grep -E '（vsftpd&#124;multi&#124;cron
-rw-r--r--. 1 root root  284  7月 30  2014 crond.service
-rw-r--r--. 1 root root  567  3月  6 06:51 multipathd.service
-rw-r--r--. 1 root root  524  3月  6 13:48 multi-user.target
drwxr-xr-x. 2 root root 4096  5月  4 17:52 multi-user.target.wants
lrwxrwxrwx. 1 root root   17  5月  4 17:52 runlevel2.target -&gt; multi-user.target
lrwxrwxrwx. 1 root root   17  5月  4 17:52 runlevel3.target -&gt; multi-user.target
lrwxrwxrwx. 1 root root   17  5月  4 17:52 runlevel4.target -&gt; multi-user.target
-rw-r--r--. 1 root root  171  6月 10  2014 vsftpd.service
-rw-r--r--. 1 root root  184  6月 10  2014 vsftpd@.service
-rw-r--r--. 1 root root   89  6月 10  2014 vsftpd.target
# 比较重要的是上头提供的那三行特殊字体的部份！
```

所以我们可以知道 vsftpd 与 crond 其实算是系统服务 （service），而 multi-user 要算是执行环境相关的类型 （target type）。根据这些扩展名的类型， 我们大概可以找到几种比较常见的 systemd 的服务类型如下：

扩展名	主要服务功能
.service	一般服务类型 （service unit）：主要是系统服务，包括服务器本身所需要的本机服务以及网络服务都是！比较经常被使用到的服务大多是这种类型！ 所以，这也是最常见的类型了！
.socket	内部程序数据交换的插槽服务 （socket unit）：主要是 IPC （Inter-process communication） 的传输讯息插槽档 （socket file） 功能。 这种类型的服务通常在监控讯息传递的插槽档，当有通过此插槽档传递讯息来说要链接服务时，就依据当时的状态将该用户的要求传送到对应的 daemon， 若 daemon 尚未启动，则启动该 daemon 后再传送用户的要求。使用 socket 类型的服务一般是比较不会被用到的服务，因此在开机时通常会稍微延迟启动的时间 （因为比较没有这么常用嘛！）。一般用于本机服务比较多，例如我们的图形界面很多的软件都是通过 socket 来进行本机程序数据交换的行为。 （这与早期的 xinetd 这个 super daemon 有部份的相似喔！）
.target	执行环境类型 （target unit）：其实是一群 unit 的集合，例如上面表格中谈到的 multi-user.target 其实就是一堆服务的集合～也就是说， 选择执行 multi-user.target 就是执行一堆其他 .service 或/及 .socket 之类的服务就是了！
.mount .automount	文件系统挂载相关的服务 （automount unit / mount unit）：例如来自网络的自动挂载、NFS 文件系统挂载等与文件系统相关性较高的程序管理。
.path	侦测特定文件或目录类型 （path unit）：某些服务需要侦测某些特定的目录来提供伫列服务，例如最常见的打印服务，就是通过侦测打印伫列目录来启动打印功能！ 这时就得要 .path 的服务类型支持了！
.timer	循环执行的服务 （timer unit）：这个东西有点类似 anacrontab 喔！不过是由 systemd 主动提供的，比 anacrontab 更加有弹性！

其中又以 .service 的系统服务类型最常见了！因为我们一堆网络服务都是通过这种类型来设计的啊！接下来，让我们来谈谈如何管理这些服务的启动与关闭。

17.2 通过 systemctl 管理服务

基本上， systemd 这个启动服务的机制，主要是通过一只名为 systemctl 的指令来处理的！跟以前 systemV 需要 service / chkconfig / setup / init 等指令来协助不同， systemd 就是仅有 systemctl 这个指令来处理而已呦！所以全部的行为都得要使用 systemctl 的意思啦！有没有很难？其实习惯了之后， 鸟哥是觉得 systemctl 还挺好用的！ ^_^

17.2.1 通过 systemctl 管理单一服务 （service unit） 的启动/开机启动与观察状态

在开始这个小节之前，鸟哥要先来跟大家报告一下，那就是：一般来说，服务的启动有两个阶段，一个是“开机的时候设置要不要启动这个服务”， 以及“你现在要不要启动这个服务”，这两者之间有很大的差异喔！举个例子来说，假如我们现在要“立刻取消 atd 这个服务”时，正规的方法 （不要用 kill） 要怎么处理？

```
[root@study ~]# systemctl [command] [unit]
command 主要有：
start      ：立刻启动后面接的 unit
stop       ：立刻关闭后面接的 unit
restart    ：立刻关闭后启动后面接的 unit，亦即执行 stop 再 start 的意思
reload     ：不关闭后面接的 unit 的情况下，重新载入配置文件，让设置生效
enable     ：设置下次开机时，后面接的 unit 会被启动
disable    ：设置下次开机时，后面接的 unit 不会被启动
status     ：目前后面接的这个 unit 的状态，会列出有没有正在执行、开机默认执行否、登录等信息等！
is-active  ：目前有没有正在运行中
is-enable  ：开机时有没有默认要启用这个 unit

范例一：看看目前 atd 这个服务的状态为何？
[root@study ~]# systemctl status atd.service
atd.service - Job spooling tools
   Loaded: loaded （/usr/lib/systemd/system/atd.service; enabled）
   Active: active （running） since Mon 2015-08-10 19:17:09 CST; 5h 42min ago
 Main PID: 1350 （atd）
   CGroup: /system.slice/atd.service
           └─1350 /usr/sbin/atd -f

Aug 10 19:17:09 study.centos.vbird systemd[1]: Started Job spooling tools.
# 重点在第二、三行喔～
# Loaded：这行在说明，开机的时候这个 unit 会不会启动，enabled 为开机启动，disabled 开机不会启动
# Active：现在这个 unit 的状态是正在执行 （running） 或没有执行 （dead）
# 后面几行则是说明这个 unit 程序的 PID 状态以及最后一行显示这个服务的登录文件信息！
# 登录文件信息格式为：“时间” “讯息发送主机” “哪一个服务的讯息” “实际讯息内容”
# 所以上面的显示讯息是：这个 atd 默认开机就启动，而且现在正在运行的意思！

范例二：正常关闭这个 atd 服务
[root@study ~]# systemctl stop atd.service
[root@study ~]# systemctl status atd.service
atd.service - Job spooling tools
   Loaded: loaded （/usr/lib/systemd/system/atd.service; enabled）
   Active: inactive （dead） since Tue 2015-08-11 01:04:55 CST; 4s ago
  Process: 1350 ExecStart=/usr/sbin/atd -f $OPTS （code=exited, status=0/SUCCESS）
 Main PID: 1350 （code=exited, status=0/SUCCESS）

Aug 10 19:17:09 study.centos.vbird systemd[1]: Started Job spooling tools.
Aug 11 01:04:55 study.centos.vbird systemd[1]: Stopping Job spooling tools...
Aug 11 01:04:55 study.centos.vbird systemd[1]: Stopped Job spooling tools.
# 目前这个 unit 下次开机还是会启动，但是现在是没在运行的状态中！同时，
# 最后两行为新增加的登录讯息，告诉我们目前的系统状态喔！
```

上面的范例中，我们已经关掉了 atd 啰！这样作才是对的！不应该使用 kill 的方式来关掉一个正常的服务喔！否则 systemctl 会无法继续监控该服务的！ 那就比较麻烦。而使用 systemtctl status atd 的输出结果中，第 2, 3 两行很重要～因为那个是告知我们该 unit 下次开机会不会默认启动，以及目前启动的状态！ 相当重要！最下面是这个 unit 的登录文件～如果你的这个 unit 曾经出错过，观察这个地方也是相当重要的！

那么现在问个问题，你的 atd 现在是关闭的，未来重新开机后，这个服务会不会再次的启动呢？答案是？当然会！ 因为上面出现的第二行中，它是 enabled 的啊！这样理解所谓的“现在的状态”跟“开机时默认的状态”两者的差异了吗？

好！再回到 systemctl status atd.service 的第三行，不是有个 Active 的 daemon 现在状态吗？除了 running 跟 dead 之外， 有没有其他的状态呢？有的～基本上有几个常见的状态：

- active （running）：正有一只或多只程序正在系统中执行的意思，举例来说，正在执行中的 vsftpd 就是这种模式。
- active （exited）：仅执行一次就正常结束的服务，目前并没有任何程序在系统中执行。

举例来说，开机或者是挂载时才会进行一次的 quotaon 功能，就是这种模式！ quotaon 不须一直执行～只须执行一次之后，就交给文件系统去自行处理啰！通常用 bash shell 写的小型服务，大多是属于这种类型 （无须常驻内存）。

- active （waiting）：正在执行当中，不过还再等待其他的事件才能继续处理。举例来说，打印的伫列相关服务就是这种状态！ 虽然正在启动中，不过，也需要真的有伫列进来 （打印工作） 这样他才会继续唤醒打印机服务来进行下一步打印的功能。
- inactive：这个服务目前没有运行的意思。

既然 daemon 目前的状态就有这么多种了，那么 daemon 的默认状态有没有可能除了 enable/disable 之外，还有其他的情况呢？当然有！

- enabled：这个 daemon 将在开机时被执行
- disabled：这个 daemon 在开机时不会被执行
- static：这个 daemon 不可以自己启动 （enable 不可），不过可能会被其他的 enabled 的服务来唤醒 （相依属性的服务）
- mask：这个 daemon 无论如何都无法被启动！因为已经被强制注销 （非删除）。可通过 systemctl unmask 方式改回原本状态

- 服务启动/关闭与观察的练习

问题：找到系统中名为 chronyd 的服务，观察此服务的状态，观察完毕后，将此服务设置为： 1）开机不会启动 2）现在状况是关闭的情况！回答：我们直接使用指令的方式来查询与设置看看：

```
# 1\. 观察一下状态，确认是否为关闭/未启动呢？
[root@study ~]# systemctl status chronyd.service
hronyd.service - NTP client/server
   Loaded: loaded （/usr/lib/systemd/system/chronyd.service; enabled）
   Active: active （running） since Mon 2015-08-10 19:17:07 CST; 24h ago
.....（下面省略）.....

# 2\. 由上面知道目前是启动的，因此立刻将他关闭，同时开机不会启动才行！
[root@study ~]# systemctl stop chronyd.service
[root@study ~]# systemctl disable chronyd.service
rm '/etc/systemd/system/multi-user.target.wants/chronyd.service'
# 看得很清楚～其实就是从 /etc/systemd/system 下面删除一条链接文件而已～

[root@study ~]# systemctl status chronyd.service
chronyd.service - NTP client/server
   Loaded: loaded （/usr/lib/systemd/system/chronyd.service; disabled）
   Active: inactive （dead）
# 如此则将 chronyd 这个服务完整的关闭了！
```

上面是一个很简单的练习，你先不要知道 chronyd 是啥东西，只要知道通过这个方式，可以将一个服务关闭就是了！好！那再来一个练习， 看看有没有问题呢？

问题：因为我根本没有打印机安装在服务器上，目前也没有网络打印机，因此我想要将 cups 服务整个关闭，是否可以呢？回答：同样的，眼见为凭，我们就动手作看看：

```
# 1\. 先看看 cups 的服务是开还是关？
[root@study ~]# systemctl status cups.service
cups.service - CUPS Printing Service
   Loaded: loaded （/usr/lib/systemd/system/cups.service; enabled）
   Active: inactive （dead） since Tue 2015-08-11 19:19:20 CST; 3h 29min ago
# 有趣得很！竟然是 enable 但是却是 inactive 耶！相当特别！

# 2\. 那就直接关闭，同时确认没有启动喔！
[root@study ~]# systemctl stop    cups.service
[root@study ~]# systemctl disable cups.service
rm '/etc/systemd/system/multi-user.target.wants/cups.path'
rm '/etc/systemd/system/sockets.target.wants/cups.socket'
rm '/etc/systemd/system/printer.target.wants/cups.service'
# 也是非常特别！竟然一口气取消掉三个链接文件！也就是说，这三个文件可能是有相依性的问题喔！

[root@study ~]# netstat -tlunp &#124; grep cups
# 现在应该不会出现任何数据！因为根本没有 cups 的任务在执行当中～所以不会有 port 产生

# 3\. 尝试启动 cups.socket 监听用户端的需求喔！
[root@study ~]# systemctl start cups.socket
[root@study ~]# systemctl status cups.service cups.socket cups.path
cups.service - CUPS Printing Service
   Loaded: loaded （/usr/lib/systemd/system/cups.service; disabled）
   Active: inactive （dead） since Tue 2015-08-11 22:57:50 CST; 3min 41s ago
cups.socket - CUPS Printing Service Sockets
   Loaded: loaded （/usr/lib/systemd/system/cups.socket; disabled）
   Active: active （listening） since Tue 2015-08-11 22:56:14 CST; 5min ago
cups.path - CUPS Printer Service Spool
   Loaded: loaded （/usr/lib/systemd/system/cups.path; disabled）
   Active: inactive （dead）
# 确定仅有 cups.socket 在启动，其他的并没有启动的状态！

# 4\. 尝试使用 lp 这个指令来打印看看？
[root@study ~]# echo "testing" &#124; lp
lp: Error - no default destination available. # 实际上就是没有打印机！所以有错误也没关系！

[root@study ~]# systemctl status cups.service
cups.service - CUPS Printing Service
   Loaded: loaded （/usr/lib/systemd/system/cups.service; disabled）
   Active: active （running） since Tue 2015-08-11 23:03:18 CST; 34s ago
[root@study ~]# netstat -tlunp &#124; grep cups
tcp        0      0 127.0.0.1:631     0.0.0.0:*    LISTEN     25881/cupsd
tcp6       0      0 ::1:631           :::*         LISTEN     25881/cupsd
# 见鬼！竟然 cups 自动被启动了！明明我们都没有驱动他啊！怎么回事啊？
```

上面这个范例的练习在让您了解一下，很多服务彼此之间是有相依性的！cups 是一种打印服务，这个打印服务会启用 port 631 来提供网络打印机的打印功能。 但是其实我们无须一直启动 631 端口吧？因此，多了一个名为 cups.socket 的服务，这个服务可以在“用户有需要打印时，才会主动唤醒 cups.service ”的意思！ 因此，如果你仅是 disable/stop cups.service 而忘记了其他两个服务的话，那么当有用户向其他两个 cups.path, cups.socket 提出要求时，cups.service 就会被唤醒！所以，你关掉也没用！

- 强迫服务注销 （mask） 的练习

比较正规的作法是，要关闭 cups.service 时，连同其他两个会唤醒 service 的 cups.socket 与 cups.path 通通关闭，那就没事了！ 比较不正规的作法是，那就强迫 cups.service 注销吧！通过 mask 的方式来将这个服务注销看看！

```
# 1\. 保持刚刚的状态，关闭 cups.service，启动 cups.socket，然后注销 cups.servcie
[root@study ~]# systemctl stop cups.service
[root@study ~]# systemctl mask cups.service
ln -s '/dev/null' '/etc/systemd/system/cups.service'
# 喔耶～其实这个 mask 注销的动作，只是让启动的脚本变成空的设备而已！

[root@study ~]# systemctl status cups.service
cups.service
   Loaded: masked （/dev/null）
   Active: inactive （dead） since Tue 2015-08-11 23:14:16 CST; 52s ago

[root@study ~]# systemctl start cups.service
Failed to issue method call: Unit cups.service is masked.  # 再也无法唤醒！
```

上面的范例你可以仔细推敲一下～原来整个启动的脚本配置文件被链接到 /dev/null 这个空设备～因此，无论如何你是再也无法启动这个 cups.service 了！ 通过这个 mask 功能，你就可以不必管其他相依服务可能会启动到这个想要关闭的服务了！虽然是非正规，不过很有效！^_^

那如何取消注销呢？当然就是 unmask 即可啊！

```
[root@study ~]# systemctl unmask cups.service
rm '/etc/systemd/system/cups.service'
[root@study ~]# systemctl status cups.service
cups.service - CUPS Printing Service
   Loaded: loaded （/usr/lib/systemd/system/cups.service; disabled）
   Active: inactive （dead） since Tue 2015-08-11 23:14:16 CST; 4min 35s ago
# 好佳在有恢复正常！
```

17.2.2 通过 systemctl 观察系统上所有的服务

上一小节谈到的是单一服务的启动/关闭/观察，以及相依服务要注销的功能。那系统上面有多少的服务存在呢？这个时候就得要通过 list-units 及 list-unit-files 来观察了！ 细部的用法如下：

```
[root@study ~]# systemctl [command] [--type=TYPE] [--all]
command:
    list-units      ：依据 unit 列出目前有启动的 unit。若加上 --all 才会列出没启动的。
    list-unit-files ：依据 /usr/lib/systemd/system/ 内的文件，将所有文件列表说明。
--type=TYPE：就是之前提到的 unit type，主要有 service, socket, target 等

范例一：列出系统上面有启动的 unit
[root@study ~]# systemctl
UNIT                      LOAD   ACTIVE SUB       DESCRIPTION
proc-sys-fs-binfmt_mis... loaded active waiting   Arbitrary Executable File Formats File
sys-devices-pc...:0:1:... loaded active plugged   QEMU_HARDDISK
sys-devices-pc...0:1-0... loaded active plugged   QEMU_HARDDISK
sys-devices-pc...0:0-1... loaded active plugged   QEMU_DVD-ROM
.....（中间省略）.....
vsftpd.service            loaded active running   Vsftpd ftp daemon
.....（中间省略）.....
cups.socket               loaded failed failed    CUPS Printing Service Sockets
.....（中间省略）.....
LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

141 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
# 列出的项目中，主要的意义是：
# UNIT   ：项目的名称，包括各个 unit 的类别 （看扩展名）
# LOAD   ：开机时是否会被载入，默认 systemctl 显示的是有载入的项目而已喔！
# ACTIVE ：目前的状态，须与后续的 SUB 搭配！就是我们用 systemctl status 观察时，active 的项目！
# DESCRIPTION ：详细描述啰
# cups 比较有趣，因为刚刚被我们玩过，所以 ACTIVE 竟然是 failed 的喔！被玩死了！ ^_^
# 另外，systemctl 都不加参数，其实默认就是 list-units 的意思！

范例二：列出所有已经安装的 unit 有哪些？
[root@study ~]# systemctl list-unit-files
UNIT FILE                                   STATE
proc-sys-fs-binfmt_misc.automount           static
dev-hugepages.mount                         static
dev-mqueue.mount                            static
proc-fs-nfsd.mount                          static
.....（中间省略）.....
systemd-tmpfiles-clean.timer                static

336 unit files listed.
```

使用 systemctl list-unit-files 会将系统上所有的服务通通列出来～而不像 list-units 仅以 unit 分类作大致的说明。 至于 STATE 状态就是前两个小节谈到的开机是否会载入的那个状态项目啰！主要有 enabled / disabled / mask / static 等等。

假设我不想要知道这么多的 unit 项目，我只想要知道 service 这种类别的 daemon 而已，而且不论是否已经启动，通通要列出来！ 那该如何是好？

```
[root@study ~]# systemctl list-units --type=service --all
# 只剩下 *.service 的项目才会出现喔！

范例一：查询系统上是否有以 cpu 为名的服务？
[root@study ~]# systemctl list-units --type=service --all &#124; grep cpu
cpupower.service  loaded inactive dead    Configure CPU power related settings
# 确实有喔！可以改变 CPU 电源管理机制的服务哩！
```

17.2.3 通过 systemctl 管理不同的操作环境 （target unit）

通过上个小节我们知道系统上所有的 systemd 的 unit 观察的方式，那么可否列出跟操作界面比较有关的 target 项目呢？ 很简单啊！就这样搞一下：

```
[root@study ~]# systemctl list-units --type=target --all
UNIT                   LOAD   ACTIVE   SUB    DESCRIPTION
basic.target           loaded active   active Basic System
cryptsetup.target      loaded active   active Encrypted Volumes
emergency.target       loaded inactive dead   Emergency Mode
final.target           loaded inactive dead   Final Step
getty.target           loaded active   active Login Prompts
graphical.target       loaded active   active Graphical Interface
local-fs-pre.target    loaded active   active Local File Systems （Pre）
local-fs.target        loaded active   active Local File Systems
multi-user.target      loaded active   active Multi-User System
network-online.target  loaded inactive dead   Network is Online
network.target         loaded active   active Network
nss-user-lookup.target loaded inactive dead   User and Group Name Lookups
paths.target           loaded active   active Paths
remote-fs-pre.target   loaded active   active Remote File Systems （Pre）
remote-fs.target       loaded active   active Remote File Systems
rescue.target          loaded inactive dead   Rescue Mode
shutdown.target        loaded inactive dead   Shutdown
slices.target          loaded active   active Slices
sockets.target         loaded active   active Sockets
sound.target           loaded active   active Sound Card
swap.target            loaded active   active Swap
sysinit.target         loaded active   active System Initialization
syslog.target          not-found inactive dead   syslog.target
time-sync.target       loaded inactive dead   System Time Synchronized
timers.target          loaded active   active Timers
umount.target          loaded inactive dead   Unmount All Filesystems

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

26 loaded units listed.
To show all installed unit files use 'systemctl list-unit-files'.
```

喔！在我们的 CentOS 7.1 的默认情况下，就有 26 个 target unit 耶！而跟操作界面相关性比较高的 target 主要有下面几个：

- graphical.target：就是文字加上图形界面，这个项目已经包含了下面的 multi-user.target 项目！
- multi-user.target：纯文本模式！
- rescue.target：在无法使用 root 登陆的情况下，systemd 在开机时会多加一个额外的暂时系统，与你原本的系统无关。这时你可以取得 root 的权限来维护你的系统。 但是这是额外系统，因此可能需要动到 chroot 的方式来取得你原有的系统喔！再后续的章节我们再来谈！
- emergency.target：紧急处理系统的错误，还是需要使用 root 登陆的情况，在无法使用 rescue.target 时，可以尝试使用这种模式！
- shutdown.target：就是关机的流程。
- getty.target：可以设置你需要几个 tty 之类的，如果想要降低 tty 的项目，可以修改这个东西的配置文件！

正常的模式是 multi-user.target 以及 graphical.target 两个，救援方面的模式主要是 rescue.target 以及更严重的 emergency.target。 如果要修改可提供登陆的 tty 数量，则修改 getty.target 项目。基本上，我们最常使用的当然就是 multi-user 以及 graphical 啰！ 那么我如何知道目前的模式是哪一种？又得要如何修改呢？下面来玩一玩吧！

```
[root@study ~]# systemctl [command] [unit.target]
选项与参数：
command:
    get-default ：取得目前的 target
    set-default ：设置后面接的 target 成为默认的操作模式
    isolate     ：切换到后面接的模式

范例一：我们的测试机器默认是图形界面，先观察是否真为图形模式，再将默认模式转为文字界面
[root@study ~]# systemctl get-default
graphical.target  # 果然是图形界面喔！

[root@study ~]# systemctl set-default multi-user.target
[root@study ~]# systemctl get-default
multi-user.target

范例二：在不重新开机的情况下，将目前的操作环境改为纯文本模式，关掉图形界面
[root@study ~]# systemctl isolate multi-user.target

范例三：若需要重新取得图形界面呢？
[root@study ~]# systemctl isolate graphical.target
```

要注意，改变 graphical.target 以及 multi-user.target 是通过 isolate 来处理的！鸟哥刚刚接触到 systemd 的时候，在 multi-user.target 环境下转成 graphical.target 时， 可以通过 systemctl start graphical.target 喔！然后鸟哥就以为关闭图形界面即可回到 multi-user.target 的！但使用 systemctl stop graphical.target 却完全不理鸟哥～这才发现错了...在 service 部份用 start/stop/restart 才对，在 target 项目则请使用 isolate （隔离不同的操作模式） 才对！

在正常的切换情况下，使用上述 isolate 的方式即可。不过为了方便起见， systemd 也提供了数个简单的指令给我们切换操作模式之用喔！ 大致上如下所示：

```
[root@study ~]# systemctl poweroff  系统关机
[root@study ~]# systemctl reboot    重新开机
[root@study ~]# systemctl suspend   进入暂停模式
[root@study ~]# systemctl hibernate 进入休眠模式
[root@study ~]# systemctl rescue    强制进入救援模式
[root@study ~]# systemctl emergency 强制进入紧急救援模式
```

关机、重新开机、救援与紧急模式这没啥问题，那么什么是暂停与休眠模式呢？

- suspend：暂停模式会将系统的状态数据保存到内存中，然后关闭掉大部分的系统硬件，当然，并没有实际关机喔！ 当使用者按下唤醒机器的按钮，系统数据会重内存中回复，然后重新驱动被大部分关闭的硬件，就开始正常运行！唤醒的速度较快。
- hibernate：休眠模式则是将系统状态保存到硬盘当中，保存完毕后，将计算机关机。当使用者尝试唤醒系统时，系统会开始正常运行， 然后将保存在硬盘中的系统状态恢复回来。因为数据是由硬盘读出，因此唤醒的性能与你的硬盘速度有关。

17.2.4 通过 systemctl 分析各服务之间的相依性

我们在本章一开始谈到 systemd 的时候就有谈到相依性的问题克服，那么，如何追踪某一个 unit 的相依性呢？ 举例来说好了，我们怎么知道 graphical.target 会用到 multi-user.target 呢？那 graphical.target 下面还有哪些东西呢？ 下面我们就来谈一谈：

```
[root@study ~]# systemctl list-dependencies [unit] [--reverse]
选项与参数：
--reverse ：反向追踪谁使用这个 unit 的意思！

范例一：列出目前的 target 环境下，用到什么特别的 unit
[root@study ~]# systemctl get-default
multi-user.target

[root@study ~]# systemctl list-dependencies
default.target
├─abrt-ccpp.service
├─abrt-oops.service
├─vsftpd.service
├─basic.target
│ ├─alsa-restore.service
│ ├─alsa-state.service
.....（中间省略）.....
│ ├─sockets.target
│ │ ├─avahi-daemon.socket
│ │ ├─dbus.socket
.....（中间省略）.....
│ ├─sysinit.target
│ │ ├─dev-hugepages.mount
│ │ ├─dev-mqueue.mount
.....（中间省略）.....
│ └─timers.target
│   └─systemd-tmpfiles-clean.timer
├─getty.target
│ └─getty@tty1.service
└─remote-fs.target
```

因为我们前一小节的练习将默认的操作模式变成 multi-user.target 了，因此这边使用 list-dependencies 时，所列出的 default.target 其实是 multi-user.target 的内容啦！根据线条连线的流程，我们也能够知道， multi-user.target 其实还会用到 basic.target + getty.target + remote-fs.target 三大项目， 而 basic.target 又用到了 sockets.target + sysinit.target + timers.target... 等一堆～所以啰，从这边就能够清楚的查询到每种 target 模式下面还有的相依模式。 那么如果要查出谁会用到 multi-user.target 呢？就这么作！

```
[root@study ~]# systemctl list-dependencies --reverse
default.target
└─graphical.target
```

reverse 本来就是反向的意思，所以加上这个选项，代表“谁还会用到我的服务”的意思～所以看得出来， multi-user.target 主要是被 graphical.target 所使用喔！ 好～那再来， graphical.target 又使用了多少的服务呢？可以这样看：

```
[root@study ~]# systemctl list-dependencies graphical.target
graphical.target
├─accounts-daemon.service
├─gdm.service
├─network.service
├─rtkit-daemon.service
├─systemd-update-utmp-runlevel.service
└─multi-user.target
  ├─abrt-ccpp.service
  ├─abrt-oops.service
.....（下面省略）.....
```

所以可以看得出来，graphical.target 就是在 multi-user.target 下面再加上 accounts-daemon, gdm, network, rtkit-deamon, systemd-update-utmp-runlevel 等服务而已！ 这样会看了吗？了解 daemon 之间的相关性也是很重要的喔！出问题时，可以找到正确的服务相依流程！

17.2.5 与 systemd 的 daemon 运行过程相关的目录简介

我们在前几小节曾经谈过比较重要的 systemd 启动脚本配置文件在 /usr/lib/systemd/system/, /etc/systemd/system/ 目录下，那还有哪些目录跟系统的 daemon 运行有关呢？ 基本上是这样的：

- /usr/lib/systemd/system/： 使用 CentOS 官方提供的软件安装后，默认的启动脚本配置文件都放在这里，这里的数据尽量不要修改～ 要修改时，请到 /etc/systemd/system 下面修改较佳！
- /run/systemd/system/： 系统执行过程中所产生的服务脚本，这些脚本的优先序要比 /usr/lib/systemd/system/ 高！
- /etc/systemd/system/： 管理员依据主机系统的需求所创建的执行脚本，其实这个目录有点像以前 /etc/rc.d/rc5.d/Sxx 之类的功能！执行优先序又比 /run/systemd/system/ 高喔！
- /etc/sysconfig/*： 几乎所有的服务都会将初始化的一些选项设置写入到这个目录下，举例来说，mandb 所要更新的 man page 索引中，需要加入的参数就写入到此目录下的 man-db 当中喔！而网络的设置则写在 /etc/sysconfig/network-scripts/ 这个目录内。所以，这个目录内的文件也是挺重要的；
- /var/lib/： 一些会产生数据的服务都会将他的数据写入到 /var/lib/ 目录中。举例来说，数据库管理系统 Mariadb 的数据库默认就是写入 /var/lib/mysql/ 这个目录下啦！
- /run/： 放置了好多 daemon 的暂存盘，包括 lock file 以及 PID file 等等。

我们知道 systemd 里头有很多的本机会用到的 socket 服务，里头可能会产生很多的 socket file ～那你怎么知道这些 socket file 放置在哪里呢？ 很简单！还是通过 systemctl 来管理！

```
[root@study ~]# systemctl list-sockets
LISTEN                          UNIT                         ACTIVATES
/dev/initctl                    systemd-initctl.socket       systemd-initctl.service
/dev/log                        systemd-journald.socket      systemd-journald.service
/run/dmeventd-client            dm-event.socket              dm-event.service
/run/dmeventd-server            dm-event.socket              dm-event.service
/run/lvm/lvmetad.socket         lvm2-lvmetad.socket          lvm2-lvmetad.service
/run/systemd/journal/socket     systemd-journald.socket      systemd-journald.service
/run/systemd/journal/stdout     systemd-journald.socket      systemd-journald.service
/run/systemd/shutdownd          systemd-shutdownd.socket     systemd-shutdownd.service
/run/udev/control               systemd-udevd-control.socket systemd-udevd.service
/var/run/avahi-daemon/socket    avahi-daemon.socket          avahi-daemon.service
/var/run/cups/cups.sock         cups.socket                  cups.service
/var/run/dbus/system_bus_socket dbus.socket                  dbus.service
/var/run/rpcbind.sock           rpcbind.socket               rpcbind.service
@ISCSIADM_ABSTRACT_NAMESPACE    iscsid.socket                iscsid.service
@ISCSID_UIP_ABSTRACT_NAMESPACE  iscsiuio.socket              iscsiuio.service
kobject-uevent 1                systemd-udevd-kernel.socket  systemd-udevd.service

16 sockets listed.
Pass --all to see loaded but inactive sockets, too.
```

这样很清楚的就能够知道正在监听本机服务需求的 socket file 所在的文件名位置啰！

- 网络服务与端口对应简介

从第十六章与前一小节对服务的说明后，你应该要知道的是， 系统所有的功能都是某些程序所提供的，而程序则是通过触发程序而产生的。同样的，系统提供的网络服务当然也是这样的！ 只是由于网络牵涉到 TCP/IP 的概念，所以显的比较复杂一些就是了。

玩过网际网络 （Internet） 的朋友应该知道 IP 这玩意儿，大家都说 IP 就是代表你的主机在网际网络上面的“门牌号码”。 但是你的主机总是可以提供非常多的网络服务而不止一项功能而已，但我们仅有一个 IP 呢！当用户端连线过来我们的主机时， 我们主机是如何分辨不同的服务要求呢？那就是通过埠号 （port number） 啦！埠号简单的想像，他就是你家门牌上面的第几层楼！ 这个 IP 与 port 就是网际网络连线的最重要机制之一啰。我们拿下面的网址来说明：

- http://ftp.ksu.edu.tw/
- ftp://ftp.ksu.edu.tw/

有没有发现，两个网址都是指向 ftp.ksu.edu.tw 这个昆山科大的 FTP 网站，但是浏览器上面显示的结果却是不一样的？ 是啊！这是因为我们指向不同的服务嘛！一个是 http 这个 WWW 的服务，一个则是 ftp 这个文件传输服务，当然显示的结果就不同了。

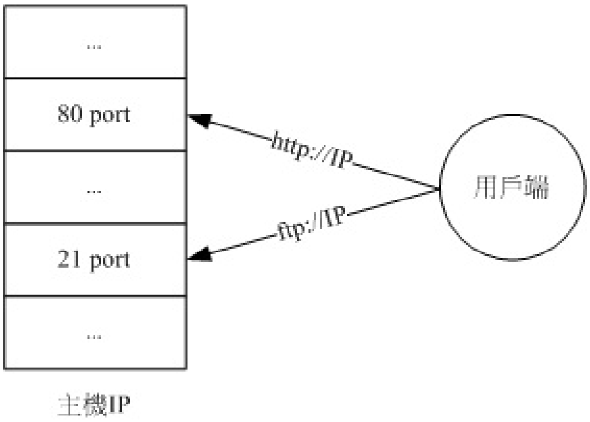


图17.2.1、port 与 daemon 的对应

事实上，为了统一整个网际网络的埠号对应服务的功能，好让所有的主机都能够使用相同的机制来提供服务与要求服务， 所以就有了“通讯协定”这玩意儿。也就是说，有些约定俗成的服务都放置在同一个埠号上面啦！举例来说， 网址列上面的 http 会让浏览器向 WWW 服务器的 80 埠号进行连线的要求！而 WWW 服务器也会将 httpd 这个软件启动在 port 80， 这样两者才能够达成连线的！

嗯！那么想一想，系统上面有没有什么设置可以让服务与埠号对应在一起呢？那就是 /etc/services 啦！

```
[root@study ~]# cat /etc/services
....（前面省略）....
ftp             21/tcp
ftp             21/udp          fsp fspd
ssh             22/tcp                          # The Secure Shell （SSH） Protocol
ssh             22/udp                          # The Secure Shell （SSH） Protocol
....（中间省略）....
http            80/tcp          www www-http    # WorldWideWeb HTTP
http            80/udp          www www-http    # HyperText Transfer Protocol
....（下面省略）....
# 这个文件的内容是以下面的方式来编排的：
# &lt;daemon name&gt;   &lt;port/封包协定&gt;   &lt;该服务的说明&gt;
```

像上面说的是，第一栏为 daemon 的名称、第二栏为该 daemon 所使用的埠号与网络数据封包协定， 封包协定主要为可靠连线的 TCP 封包以及较快速但为非连线导向的 UDP 封包。 举个例子说，那个远端连线机制使用的是 ssh 这个服务，而这个服务的使用的埠号为 22 ！就是这样啊！



Tips 请特别注意！虽然有的时候你可以借由修改 /etc/services 来更改一个服务的埠号，不过并不建议如此做， 因为很有可能会造成一些协定的错误情况！这里特此说明一番呦！（除非你要架设一个地下网站，否则的话，使用 /etc/services 原先的设置就好啦！）

17.2.6 关闭网络服务

当你第一次使用 systemctl 去观察本机服务器启动的服务时，不知道有没有吓一跳呢？怎么随随便便 CentOS 7.x 就给我启动了几乎 100 多个以上的 daemon？ 会不会有事啊？没关系啦！因为 systemd 将许多原本不被列为 daemon 的程序都纳入到 systemd 自己的管辖监测范围内，因此就多了很多 daemon 存在！ 那些大部分都属于 Linux 系统基础运行所需要的环境，没有什么特别需求的话，最好都不要更动啦！除非你自己知道自己需要什么。

除了本机服务之外，其实你一定要观察的，反而是网络服务喔！虽然网络服务默认有 SELinux 管理，不过，在鸟哥的立场上， 我还是建议非必要的网络服务就关闭他！那么什么是网络服务呢？基本上，会产生一个网络监听端口 （port） 的程序，你就可以称他是个网络服务了！ 那么如何观察网络端口？就这样追踪啊！

```
[root@study ~]# netstat -tlunp
Proto Recv-Q Send-Q Local Address   Foreign Address  State    PID/Program name
tcp        0      0 0.0.0.0:22      0.0.0.0:*        LISTEN   1340/sshd
tcp        0      0 127.0.0.1:25    0.0.0.0:*        LISTEN   2387/master
tcp6       0      0 :::555          :::*             LISTEN   29113/vsftpd
tcp6       0      0 :::22           :::*             LISTEN   1340/sshd
tcp6       0      0 ::1:25          :::*             LISTEN   2387/master
udp        0      0 0.0.0.0:5353    0.0.0.0:*                 750/avahi-daemon: r
udp        0      0 0.0.0.0:36540   0.0.0.0:*                 750/avahi-daemon: r
```

如上表所示，我们的系统上至少开了 22, 25, 555, 5353, 36540 这几个端口～而其中 5353, 36540 是由 avahi-daemon 这个东西所启动的！ 接下来我们使用 systemctl 去观察一下，到底有没有 avahi-daemon 为开头的服务呢？

```
[root@study ~]# systemctl list-units --all &#124; grep avahi-daemon
avahi-daemon.service   loaded active   running   Avahi mDNS/DNS-SD Stack
avahi-daemon.socket    loaded active   running   Avahi mDNS/DNS-SD Stack Activation Socke
```

通过追查，知道这个 avahi-daemon 的目的是在区域网络进行类似网芳的搜寻，因此这个服务可以协助你在区网内随时了解随插即用的设备！ 包括笔记本电脑等，只要连上你的区网，你就能够知道谁进来了。问题是，你可能不要这个协定啊！所以，那就关闭他吧！

```
[root@study ~]# systemctl stop avahi-daemon.service
[root@study ~]# systemctl stop avahi-daemon.socket
[root@study ~]# systemctl disable avahi-daemon.service avahi-daemon.socket
[root@study ~]# netstat -tlunp
Proto Recv-Q Send-Q Local Address   Foreign Address  State    PID/Program name
tcp        0      0 0.0.0.0:22      0.0.0.0:*        LISTEN   1340/sshd
tcp        0      0 127.0.0.1:25    0.0.0.0:*        LISTEN   2387/master
tcp6       0      0 :::555          :::*             LISTEN   29113/vsftpd
tcp6       0      0 :::22           :::*             LISTEN   1340/sshd
tcp6       0      0 ::1:25          :::*             LISTEN   2387/master
```

一般来说，你的本机服务器至少需要 25 号端口，而 22 号端口则最好加上防火墙来管理远端连线登陆比较妥当～因此，上面的端口中， 除了 555 是我们上一章因为测试而产生的之外，这样的系统能够被爬墙的机会已经少很多了！ ^_^！OK！现在如果你的系统里面有一堆网络端口在监听， 而你根本不知道那是干麻用的，鸟哥建议你，现在就通过上面的方式，关闭他吧！

17.3 systemctl 针对 service 类型的配置文件

以前，我们如果想要创建系统服务，就得要到 /etc/init.d/ 下面去创建相对应的 bash shell script 来处理。那么现在 systemd 的环境下面， 如果我们想要设置相关的服务启动环境，那应该如何处理呢？这就是本小节的任务啰！

17.3.1 systemctl 配置文件相关目录简介

现在我们知道服务的管理是通过 systemd，而 systemd 的配置文件大部分放置于 /usr/lib/systemd/system/ 目录内。但是 Red Hat 官方文件指出， 该目录的文件主要是原本软件所提供的设置，建议不要修改！而要修改的位置应该放置于 /etc/systemd/system/ 目录内。举例来说，如果你想要额外修改 vsftpd.service 的话， 他们建议要放置到哪些地方呢？

- /usr/lib/systemd/system/vsftpd.service：官方释出的默认配置文件；
- /etc/systemd/system/vsftpd.service.d/custom.conf：在 /etc/systemd/system 下面创建与配置文件相同文件名的目录，但是要加上 .d 的扩展名。然后在该目录下创建配置文件即可。另外，配置文件最好附文件名取名为 .conf 较佳！ 在这个目录下的文件会“累加其他设置”进入 /usr/lib/systemd/system/vsftpd.service 内喔！
- /etc/systemd/system/vsftpd.service.wants/*：此目录内的文件为链接文件，设置相依服务的链接。意思是启动了 vsftpd.service 之后，最好再加上这目录下面建议的服务。
- /etc/systemd/system/vsftpd.service.requires/*：此目录内的文件为链接文件，设置相依服务的链接。意思是在启动 vsftpd.service 之前，需要事先启动哪些服务的意思。

基本上，在配置文件里面你都可以自由设置相依服务的检查，并且设置加入到哪些 target 里头去。但是如果是已经存在的配置文件，或者是官方提供的配置文件， Red Hat 是建议你不要修改原设置，而是到上面提到的几个目录去进行额外的客制化设置比较好！当然，这见仁见智～如果你硬要修改原始的 /usr/lib/systemd/system 下面的配置文件，那也是 OK 没问题的！并且也能够减少许多配置文件的增加～鸟哥自己认为，这样也不错！反正，就完全是个人喜好啰～

17.3.2 systemctl 配置文件的设置项目简介

了解了配置文件的相关目录与文件之后，再来，当然得要了解一下配置文件本身的内容了！让我们先来瞧一瞧 sshd.service 的内容好了！ 原本想拿 vsftpd.service 来讲解，不过该文件的内容比较阳春，还是看一下设置项目多一些的 sshd.service 好了！

```
[root@study ~]# cat /usr/lib/systemd/system/sshd.service
[Unit]           # 这个项目与此 unit 的解释、执行服务相依性有关
Description=OpenSSH server daemon
After=network.target sshd-keygen.service
Wants=sshd-keygen.service

[Service]        # 这个项目与实际执行的指令参数有关
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]        # 这个项目说明此 unit 要挂载哪个 target 下面
WantedBy=multi-user.target
```

分析上面的配置文件，我们大概能够将整个设置分为三个部份，就是：

- [Unit]： unit 本身的说明，以及与其他相依 daemon 的设置，包括在什么服务之后才启动此 unit 之类的设置值；
- [Service], [Socket], [Timer], [Mount], [Path]..：不同的 unit type 就得要使用相对应的设置项目。我们拿的是 sshd.service 来当范本，所以这边就使用 [Service] 来设置。 这个项目内主要在规范服务启动的脚本、环境配置文件文件名、重新启动的方式等等。
- [Install]：这个项目就是将此 unit 安装到哪个 target 里面去的意思！

至于配置文件内有些设置规则还是得要说明一下：

- 设置项目通常是可以重复的，例如我可以重复设置两个 After 在配置文件中，不过，后面的设置会取代前面的喔！因此，如果你想要将设置值归零， 可以使用类似“ After= ”的设置，亦即该项目的等号后面什么都没有，就将该设置归零了 （reset）。
- 如果设置参数需要有“是/否”的项目 （布林值, boolean），你可以使用 1, yes, true, on 代表启动，用 0, no, false, off 代表关闭！随你喜好选择啰！
- 空白行、开头为 # 或 ; 的那一行，都代表注解！

每个部份里面还有很多的设置细项，我们使用一个简单的表格来说明每个项目好了！

[Unit] 部份	
设置参数	参数意义说明
Description	就是当我们使用 systemctl list-units 时，会输出给管理员看的简易说明！当然，使用 systemctl status 输出的此服务的说明，也是这个项目！
Documentation	这个项目在提供管理员能够进行进一步的文件查询的功能！提供的文件可以是如下的数据： `Documentation=http://www....` `Documentation=man:sshd（8）` `Documentation=file:/etc/ssh/sshd_config`
After	说明此 unit 是在哪个 daemon 启动之后才启动的意思！基本上仅是说明服务启动的顺序而已，并没有强制要求里头的服务一定要启动后此 unit 才能启动。 以 sshd.service 的内容为例，该文件提到 After 后面有 network.target 以及 sshd-keygen.service，但是若这两个 unit 没有启动而强制启动 sshd.service 的话， 那么 sshd.service 应该还是能够启动的！这与 Requires 的设置是有差异的喔！
Before	与 After 的意义相反，是在什么服务启动前最好启动这个服务的意思。不过这仅是规范服务启动的顺序，并非强制要求的意思。
Requires	明确的定义此 unit 需要在哪个 daemon 启动后才能够启动！就是设置相依服务啦！如果在此项设置的前导服务没有启动，那么此 unit 就不会被启动！
Wants	与 Requires 刚好相反，规范的是这个 unit 之后最好还要启动什么服务比较好的意思！不过，并没有明确的规范就是了！主要的目的是希望创建让使用者比较好操作的环境。 因此，这个 Wants 后面接的服务如果没有启动，其实不会影响到这个 unit 本身！
Conflicts	代表冲突的服务！亦即这个项目后面接的服务如果有启动，那么我们这个 unit 本身就不能启动！我们 unit 有启动，则此项目后的服务就不能启动！ 反正就是冲突性的检查啦！

接下来了解一下在 [Service] 当中有哪些项目可以使用！

[Service] 部份	
设置参数	参数意义说明
Type	说明这个 daemon 启动的方式，会影响到 ExecStart 喔！一般来说，有下面几种类型 simple：默认值，这个 daemon 主要由 ExecStart 接的指令串来启动，启动后常驻于内存中。forking：由 ExecStart 启动的程序通过 spawns 延伸出其他子程序来作为此 daemon 的主要服务。原生的父程序在启动结束后就会终止运行。 传统的 unit 服务大多属于这种项目，例如 httpd 这个 WWW 服务，当 httpd 的程序因为运行过久因此即将终结了，则 systemd 会再重新生出另一个子程序持续运行后， 再将父程序删除。据说这样的性能比较好！！oneshot：与 simple 类似，不过这个程序在工作完毕后就结束了，不会常驻在内存中。dbus：与 simple 类似，但这个 daemon 必须要在取得一个 D-Bus 的名称后，才会继续运行！因此设置这个项目时，通常也要设置 BusName= 才行！idle：与 simple 类似，意思是，要执行这个 daemon 必须要所有的工作都顺利执行完毕后才会执行。这类的 daemon 通常是开机到最后才执行即可的服务！比较重要的项目大概

	是 simple, forking 与 oneshot 了！毕竟很多服务需要子程序（forking），而有更多的动作只需要在开机的时候执行一次（oneshot），例如文件系统的检查与挂载啊等等的。
EnvironmentFile	可以指定启动脚本的环境配置文件！例如 sshd.service 的配置文件写入到 /etc/sysconfig/sshd 当中！你也可以使用 Environment= 后面接多个不同的 Shell 变量来给予设置！
ExecStart	就是实际执行此 daemon 的指令或脚本程序。你也可以使用 ExecStartPre （之前） 以及 ExecStartPost （之后） 两个设置项目来在实际启动服务前，进行额外的指令行为。 但是你得要特别注意的是，指令串仅接受“指令 参数 参数...”的格式，不能接受 <, >, >>, |, & 等特殊字符，很多的 bash 语法也不支持喔！ 所以，要使用这些特殊的字符时，最好直接写入到指令脚本里面去！不过，上述的语法也不是完全不能用，亦即，若要支持比较完整的 bash 语法，那你得要使用 Type=oneshot 才行喔！ 其他的 Type 才不能支持这些字符。
ExecStop	与 systemctl stop 的执行有关，关闭此服务时所进行的指令。
ExecReload	与 systemctl reload 有关的指令行为
Restart	当设置 Restart=1 时，则当此 daemon 服务终止后，会再次的启动此服务。举例来说，如果你在 tty2 使用文字界面登陆，操作完毕后登出，基本上，这个时候 tty2 就已经结束服务了。 但是你会看到屏幕又立刻产生一个新的 tty2 的登陆画面等待你的登陆！那就是 Restart 的功能！除非使用 systemctl 强制将此服务关闭，否则这个服务会源源不绝的一直重复产生！
RemainAfterExit	当设置为 RemainAfterExit=1 时，则当这个 daemon 所属的所有程序都终止之后，此服务会再尝试启动。这对于 Type=oneshot 的服务很有帮助！
TimeoutSec	若这个服务在启动或者是关闭时，因为某些缘故导致无法顺利“正常启动或正常结束”的情况下，则我们要等多久才进入“强制结束”的状态！
KillMode	可以是 process, control-group, none 的其中一种，如果是 process 则 daemon 终止时，只会终止主要的程序 （ExecStart 接的后面那串指令），如果是 control-group 时， 则由此 daemon 所产生的其他 control-group 的程序，也都会被关闭。如果是 none 的话，则没有程序会被关闭喔！
RestartSec	与 Restart 有点相关性，如果这个服务被关闭，然后需要重新启动时，大概要 sleep 多少时间再重新启动的意思。默认是 100ms （毫秒）。

最后，再来看看那么 Install 内还有哪些项目可用？

[Install] 部份	
设置参数	参数意义说明
WantedBy	这个设置后面接的大部分是 *.target unit ！意思是，这个 unit 本身是附挂在哪一个 target unit 下面的！一般来说，大多的服务性质的 unit 都是附挂在 multi-user.target 下面！
Also	当目前这个 unit 本身被 enable 时，Also 后面接的 unit 也请 enable 的意思！也就是具有相依性的服务可以写在这里呢！
Alias	进行一个链接的别名的意思！当 systemctl enable 相关的服务时，则此服务会进行链接文件的创建！以 multi-user.target 为例，这个家伙是用来作为默认操作环境 default.target 的规划， 因此当你设置用成 default.target 时，这个 /etc/systemd/system/default.target 就会链接到 /usr/lib/systemd/system/multi-user.target 啰！

大致的项目就有这些，接下来让我们根据上面这些数据来进行一些简易的操作吧！

17.3.3 两个 vsftpd 运行的实例

我们在上一章将 vsftpd 的 port 改成 555 号了。不过，因为某些原因，所以你可能需要使用到两个端口，分别是正常的 21 以及特殊的 555 ！ 这两个 port 都启用的情况下，你可能就得要使用到两个配置文件以及两个启动脚本设置了！现在假设是这样：

- 默认的 port 21：使用 /etc/vsftpd/vsftpd.conf 配置文件，以及 /usr/lib/systemd/system/vsftpd.service 设置脚本；
- 特殊的 port 555：使用 /etc/vsftpd/vsftpd2.conf 配置文件，以及 /etc/systemd/system/vsftpd2.service 设置脚本。

我们可以这样作：

```
# 1\. 先创建好所需要的配置文件
[root@study ~]# cd /etc/vsftpd
[root@study vsftpd]# cp vsftpd.conf vsftpd2.conf
[root@study vsftpd]# vim vsftpd.conf
#listen_port=555

[root@study vsftpd]# diff vsftpd.conf vsftpd2.conf
128c128
&lt; #listen_port=555
---
&gt; listen_port=555
# 注意这两个配置文件的差别喔！只有这一行不同而已！

# 2\. 开始处理启动脚本设置
[root@study vsftpd]# cd /etc/systemd/system
[root@study system]# cp /usr/lib/systemd/system/vsftpd.service vsftpd2.service
[root@study system]# vim vsftpd2.service
[Unit]
Description=Vsftpd second ftp daemon
After=network.target

[Service]
Type=forking
ExecStart=/usr/sbin/vsftpd /etc/vsftpd/vsftpd2.conf

[Install]
WantedBy=multi-user.target
# 重点在改了 vsftpd2.conf 这个配置文件喔！

# 3\. 重新载入 systemd 的脚本配置文件内容
[root@study system]# systemctl daemon-reload
[root@study system]# systemctl list-unit-files --all &#124; grep vsftpd
vsftpd.service                               enabled
vsftpd2.service                              disabled
vsftpd@.service                              disabled
vsftpd.target                                disabled

[root@study system]# systemctl status vsftpd2.service
vsftpd2.service - Vsftpd second ftp daemon
   Loaded: loaded （/etc/systemd/system/vsftpd2.service; disabled）
   Active: inactive （dead）

[root@study system]# systemctl restart vsftpd.service vsftpd2.service
[root@study system]# systemctl enable  vsftpd.service vsftpd2.service
[root@study system]# systemctl status  vsftpd.service vsftpd2.service
vsftpd.service - Vsftpd ftp daemon
   Loaded: loaded （/usr/lib/systemd/system/vsftpd.service; enabled）
   Active: active （running） since Wed 2015-08-12 22:00:17 CST; 35s ago
 Main PID: 12670 （vsftpd）
   CGroup: /system.slice/vsftpd.service
           └─12670 /usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf

Aug 12 22:00:17 study.centos.vbird systemd[1]: Started Vsftpd ftp daemon.

vsftpd2.service - Vsftpd second ftp daemon
   Loaded: loaded （/etc/systemd/system/vsftpd2.service; enabled）
   Active: active （running） since Wed 2015-08-12 22:00:17 CST; 35s ago
 Main PID: 12672 （vsftpd）
   CGroup: /system.slice/vsftpd2.service
           └─12672 /usr/sbin/vsftpd /etc/vsftpd/vsftpd2.conf

[root@study system]# netstat -tlnp
Active Internet connections （only servers）
Proto Recv-Q Send-Q Local Address  Foreign Address   State    PID/Program name
tcp        0      0 0.0.0.0:22     0.0.0.0:*         LISTEN   1340/sshd
tcp        0      0 127.0.0.1:25   0.0.0.0:*         LISTEN   2387/master
tcp6       0      0 :::555         :::*              LISTEN   12672/vsftpd
tcp6       0      0 :::21          :::*              LISTEN   12670/vsftpd
tcp6       0      0 :::22          :::*              LISTEN   1340/sshd
tcp6       0      0 ::1:25         :::*              LISTEN   2387/master
```

很简单的将你的 systemd 所管理的 vsftpd 做了另一个服务！未来如果有相同的需求，同样的方法作一遍即可！

17.3.4 多重的重复设置方式：以 getty 为例

我们的 CentOS 7 开机完成后，不是说有 6 个终端机可以使用吗？就是那个 tty1~tty6 的啊！那个东西是由 agetty 这个指令达成的。 OK！那么这个终端机的功能又是从哪个项目所提供的呢？其实，那个东东涉及很多层面，主要管理的是 getty.target 这个 target unit ， 不过，实际产生 tty1~tty6 的则是由 getty@.service 所提供的！咦！那个 @ 是啥东西？

先来查阅一下 /usr/lib/systemd/system/getty@.service 的内容好了：

```
[root@study ~]# cat //usr/lib/systemd/system/getty@.service
[Unit]
Description=Getty on %I
Documentation=man:agetty（8） man:systemd-getty-generator（8）
Documentation=http://0pointer.de/blog/projects/serial-console.html
After=systemd-user-sessions.service plymouth-quit-wait.service
After=rc-local.service
Before=getty.target
ConditionPathExists=/dev/tty0

[Service]
ExecStart=-/sbin/agetty --noclear %I $TERM
Type=idle
Restart=always
RestartSec=0
UtmpIdentifier=%I
TTYPath=/dev/%I
TTYReset=yes
TTYVHangup=yes
TTYVTDisallocate=yes
KillMode=process
IgnoreSIGPIPE=no
SendSIGHUP=yes

[Install]
WantedBy=getty.target
```

比较重要的当然就是 ExecStart 项目啰！那么我们去 man agetty 时，发现到它的语法应该是“ agetty --noclear tty1 ”之类的字样， 因此，我们如果要启动六个 tty 的时候，基本上应该要有六个启动配置文件。亦即是可能会用到 getty1.service, getty2.service...getty6.service 才对！哇！这样控管很麻烦啊～所以，才会出现这个 @ 的项目啦！咦！这个 @ 到底怎么回事呢？我们先来看看 getty@.service 的上游，亦即是 getty.target 这个东西的内容好了！

```
[root@study ~]# systemctl show getty.target
# 那个 show 的指令可以将 getty.target 的默认设置值也取出来显示！
Names=getty.target
Wants=getty@tty1.service
WantedBy=multi-user.target
Conflicts=shutdown.target
Before=multi-user.target
After=getty@tty1.service getty@tty2.service getty@tty3.service getty@tty4.service
  getty@tty6.service getty@tty5.service
.....（后面省略）.....
```

你会发现，咦！怎么会多出六个怪异的 service 呢？我们拿 getty@tty1.service 来说明一下好了！当我们执行完 getty.target 之后， 他会持续要求 getty@tty1.service 等六个服务继续启动。那我们的 systemd 就会这么作：

- 先看 /usr/lib/systemd/system/, /etc/systemd/system/ 有没有 getty@tty1.service 的设置，若有就执行，若没有则执行下一步；
- 找 getty@.service 的设置，若有则将 @ 后面的数据带入成 %I 的变量，进入 getty@.service 执行！

这也就是说，其实 getty@tty1.service 实际上是不存在的！他主要是通过 getty@.service 来执行～也就是说， getty@.service 的目的是为了要简化多个执行的启动设置， 他的命名方式是这样的：

```
原始文件：执行服务名称@.service
可执行文件案：执行服务名称@范例名称.service
```

因此当有范例名称带入时，则会有一个新的服务名称产生出来！你再回头看看 getty@.service 的启动脚本：

```
ExecStart=-/sbin/agetty --noclear %I $TERM
```

上表中那个 %I 指的就是“范例名称”！根据 getty.target 的信息输出来看，getty@tty1.service 的 %I 就是 tty1 啰！因此执行脚本就会变成“ /sbin/agetty --noclear tty1 ”！ 所以我们才有办法以一个配置文件来启动多个 tty1 给用户登陆啰！

- 将 tty 的数量由 6 个降低到 4 个

现在你应该要感到困扰的是，那么“ 6 个 tty 是谁规定的”为什么不是 5 个还是 7 个？这是因为 systemd 的登陆配置文件 /etc/systemd/logind.conf 里面规范的啦！ 假如你想要让 tty 数量降低到剩下 4 个的话，那么可以这样实验看看：

```
# 1\. 修改默认的 logind.conf 内容，将原本 6 个虚拟终端机改成 4 个
[root@study ~]# vim /etc/systemd/logind.conf
[Login]
NAutoVTs=4
ReserveVT=0
# 原本是 6 个而且还注解，请取消注解，然后改成 4 吧！

# 2\. 关闭不小心启动的 tty5, tty6 并重新启动 getty.target 啰！
[root@study ~]# systemctl stop getty@tty5.service
[root@study ~]# systemctl stop getty@tty6.service
[root@study ~]# systemctl restart systemd-logind.service
```

现在你再到桌面环境下，按下 [ctrl]+[alt]+[F1]~[F6] 就会发现，只剩下四个可用的 tty 啰！后面的 tty5, tty6 已经被放弃了！不再被启动喔！ 好！那么我暂时需要启动 tty8 时，又该如何处理呢？需要重新创建一个脚本吗？不需要啦！可以这样作！

```
[root@study ~]# systemctl start getty@tty8.service
```

无须额外创建其他的启动服务配置文件喔！

- 暂时新增 vsftpd 到 2121 端口

不知道你有没有发现，其实在 /usr/lib/systemd/system 下面还有个特别的 vsftpd@.service 喔！来看看他的内容：

```
[root@study ~]# cat /usr/lib/systemd/system/vsftpd@.service
[Unit]
Description=Vsftpd ftp daemon
After=network.target
PartOf=vsftpd.target

[Service]
Type=forking
ExecStart=/usr/sbin/vsftpd /etc/vsftpd/%i.conf

[Install]
WantedBy=vsftpd.target
```

根据前面 getty@.service 的说明，我们知道在启动的脚本设置当中， %i 或 %I 就是代表 @ 后面接的范例文件名的意思！ 那我能不能创建 vsftpd3.conf 文件，然后通过该文件来启动新的服务呢？就来玩玩看！

```
# 1\. 根据 vsftpd@.service 的建议，于 /etc/vsftpd/ 下面先创建新的配置文件
[root@study ~]# cd /etc/vsftpd
[root@study vsftpd]# cp vsftpd.conf vsftpd3.conf
[root@study vsftpd]# vim vsftpd3.conf
listen_port=2121

# 2\. 暂时启动这个服务，不要永久启动他！
[root@study vsftpd]# systemctl start vsftpd@vsftpd3.service
[root@study vsftpd]# systemctl status vsftpd@vsftpd3.service
vsftpd@vsftpd3.service - Vsftpd ftp daemon
   Loaded: loaded （/usr/lib/systemd/system/vsftpd@.service; disabled）
   Active: active （running） since Thu 2015-08-13 01:34:05 CST; 5s ago

[root@study vsftpd]# netstat -tlnp
Active Internet connections （only servers）
Proto Recv-Q Send-Q Local Address  Foreign Address  State    PID/Program name
tcp6       0      0 :::2121        :::*             LISTEN   16404/vsftpd
tcp6       0      0 :::555         :::*             LISTEN   12672/vsftpd
tcp6       0      0 :::21          :::*             LISTEN   12670/vsftpd
```

因为我们启用了 vsftpd@vsftpd3.service ，代表要使用的配置文件在 /etc/vsftpd/vsftpd3.conf 的意思！所以可以直接通过 vsftpd@.service 而无须重新设置启动脚本！ 这样是否比前几个小节的方法还要简便呢？ ^^。通过这个方式，你就可以使用到新的配置文件啰！只是你得要注意到 @ 这个东西就是了！ ^^



Tips 聪明的读者可能立刻发现一件事，为啥这次 FTP 增加了 2121 端口却不用修改 SELinux 呢？这是因为默认启动小于 1024 号码以下的端口时， 需要使用到 root 的权限，因此小于 1024 以下端口的启动较可怕。而这次范例中，我们使用 2121 端口，他对于系统的影响可能小一些 （其实一样可怕！）， 所以就忽略了 SELinux 的限制了！

17.3.5 自己的服务自己作

我们来仿真自己作一个服务吧！假设我要作一只可以备份自己系统的服务，这只脚本我放在 /backups 下面，内容有点像这样：

```
[root@study ~]# vim /backups/backup.sh
#!/bin/bash

source="/etc /home /root /var/lib /var/spool/{cron,at,mail}"
target="/backups/backup-system-$（date +%Y-%m-%d）.tar.gz"
[ ! -d /backups ] && mkdir /backups
tar -zcvf ${target} ${source} &&gt; /backups/backup.log

[root@study ~]# chmod a+x /backups/backup.sh
[root@study ~]# ll /backups/backup.sh
-rwxr-xr-x. 1 root root 220 Aug 13 01:57 /backups/backup.sh
# 记得要有可执行的权限才可以喔！
```

接下来，我们要如何设计一只名为 backup.service 的启动脚本设置呢？可以这样做喔！

```
[root@study ~]# vim /etc/systemd/system/backup.service
[Unit]
Description=backup my server
Requires=atd.service

[Service]
Type=simple
ExecStart=/bin/bash -c " echo /backups/backup.sh &#124; at now"

[Install]
WantedBy=multi-user.target
# 因为 ExecStart 里面有用到 at 这个指令，因此， atd.service 就是一定要的服务！

[root@study ~]# systemctl daemon-reload
[root@study ~]# systemctl start backup.service
[root@study ~]# systemctl status backup.service
backup.service - backup my server
   Loaded: loaded （/etc/systemd/system/backup.service; disabled）
   Active: inactive （dead）

Aug 13 07:50:31 study.centos.vbird systemd[1]: Starting backup my server...
Aug 13 07:50:31 study.centos.vbird bash[20490]: job 8 at Thu Aug 13 07:50:00 2015
Aug 13 07:50:31 study.centos.vbird systemd[1]: Started backup my server.
# 为什么 Active 是 inactive 呢？这是因为我们的服务仅是一个简单的 script 啊！
# 因此执行完毕就完毕了，不会继续存在内存中喔！
```

完成上述的动作之后，以后你都可以直接使用 systemctl start backup.service 进行系统的备份了！而且会直接丢进 atd 的管理中， 你就无须自己手动用 at 去处理这项任务了～好像还不赖喔！ ^_^

这样自己做一个服务好像也不难啊！ ^_^！自己动手玩玩看吧！

17.4 systemctl 针对 timer 的配置文件

有时候，某些服务你想要定期执行，或者是开机后执行，或者是什么服务启动多久后执行等等的。在过去，我们大概都是使用 crond 这个服务来定期处理， 不过，既然现在有一直常驻在内存当中的 systemd 这个好用的东西，加上这 systemd 有个协力服务，名为 timers.target 的家伙，这家伙可以协助定期处理各种任务！ 那么，除了 crond 之外，如何使用 systemd 内置的 time 来处理各种任务呢？这就是本小节的重点啰！

- systemd.timer 的优势

在 archlinux 的官网 wiki 上面有提到，为啥要使用 systemd.timer 呢？

- 由于所有的 systemd 的服务产生的信息都会被纪录 （log），因此比 crond 在 debug 上面要更清楚方便的多；
- 各项 timer 的工作可以跟 systemd 的服务相结合；
- 各项 timer 的工作可以跟 control group （cgroup，用来取代 /etc/secure/limit.conf 的功能） 结合，来限制该工作的资源利用

虽然还是有些弱点啦～例如 systemd 的 timer 并没有 email 通知的功能 （除非自己写一个），也没有类似 anacron 的一段时间内的随机取样功能 （random_delay）， 不过，总体来说，还是挺不错的！此外，相对于 crond 最小的单位到分， systemd 是可以到秒甚至是毫秒的单位哩！相当有趣！

- 任务需求

基本上，想要使用 systemd 的 timer 功能，你必须要有几个要件：

- 系统的 timer.target 一定要启动
- 要有个 sname.service 的服务存在 （sname 是你自己指定的名称）
- 要有个 sname.timer 的时间启动服务存在

满足上面的需求就 OK 了！有没有什么案例可以来实作看看？这样说好了，我们上个小节不是才自己做了个 backup.service 的服务吗？那么能不能将这个 backup.service 用在定期执行上面呢？好啊！那就来测试看看！

- sname.timer 的设置值

你可以到 /etc/systemd/system 下面去创建这个 *.timer 档，那这个文件的内容要项有哪些东西呢？基本设置主要有下面这些： （man systemd.timer & man systemd.time）

[Timer] 部份	
设置参数	参数意义说明
OnActiveSec	当 timers.target 启动多久之后才执行这只 unit
OnBootSec	当开机完成后多久之后才执行
OnStartupSec	当 systemd 第一次启动之后过多久才执行
OnUnitActiveSec	这个 timer 配置文件所管理的那个 unit 服务在最后一次启动后，隔多久后再执行一次的意思
OnUnitInactiveSec	这个 timer 配置文件所管理的那个 unit 服务在最后一次停止后，隔多久再执行一次的意思。
OnCalendar	使用实际时间 （非循环时间） 的方式来启动服务的意思！至于时间的格式后续再来谈。
Unit	一般来说不太需要设置，因此如同上面刚刚提到的，基本上我们设置都是 sname.server + sname.timer，那如果你的 sname 并不相同时，那在 .timer 的文件中， 就得要指定是哪一个 service unit 啰！
Persistent	当使用 OnCalendar 的设置时，指定该功能要不要持续进行的意思。通常是设置为 yes ，比较能够满足类似 anacron 的功能喔！

基本的项目仅有这些而已，在设置上其实并不困难啦！

- 使用于 OnCalendar 的时间

如果你想要从 crontab 转成这个 timer 功能的话，那么对于时间设置的格式就得要了解了解～基本上的格式如下所示：

```
语法：英文周名  YYYY-MM-DD  HH:MM:SS
范例：Thu       2015-08-13  13:40:00
```

上面谈的是基本的语法，你也可以直接使用间隔时间来处理！常用的间隔时间单位有：

- us 或 usec：微秒 （10-6 秒）
- ms 或 msec：毫秒 （10-3 秒）
- s, sec, second, seconds
- m, min, minute, minutes
- h, hr, hour, hours
- d, day, days
- w, week, weeks
- month, months
- y, year, years

常见的使用范例有：

```
隔 3 小时：            3h  或 3hr 或 3hours
隔 300 分钟过 10 秒：   10s 300m
隔 5 天又 100 分钟：    100m 5day
# 通常英文的写法，小单位写前面，大单位写后面～所以先秒、再分、再小时、再天数等～
```

此外，你也可以使用英文常用的口语化日期代表，例如 today, tomorrow 等！假设今天是 2015-08-13 13:50:00 的话，那么：

英文口语	实际的时间格式代表
now	Thu 2015-08-13 13:50:00
today	Thu 2015-08-13 00:00:00
tomorrow	Thu 2015-08-14 00:00:00
hourly	-- :00:00
daily	--* 00:00:00
weekly	Mon --* 00:00:00
monthly	--01 00:00:00
+3h10m	Thu 2015-08-13 17:00:00
2015-08-16	Sun 2015-08-16 00:00:00

- 一个循环时间运行的案例

现在假设这样：

- 开机后 2 小时开始执行一次这个 backup.service
- 自从第一次执行后，未来我每两天要执行一次 backup.service

好了，那么应该如何处理这个脚本呢？可以这样做喔！

```
[root@study ~]# vim /etc/systemd/system/backup.timer
[Unit]
Description=backup my server timer

[Timer]
OnBootSec=2hrs
OnUnitActiveSec=2days

[Install]
WantedBy=multi-user.target
# 只要这样设置就够了！储存离开吧！

[root@study ~]# systemctl daemon-reload
[root@study ~]# systemctl enable backup.timer
[root@study ~]# systemctl restart backup.timer
[root@study ~]# systemctl list-unit-files &#124; grep backup
backup.service          disabled    # 这个不需要启动！只要 enable backup.timer 即可！
backup.timer            enabled

[root@study ~]# systemctl show timers.target
ConditionTimestamp=Thu 2015-08-13 14:31:11 CST      # timer 这个 unit 启动的时间！

[root@study ~]# systemctl show backup.service
ExecMainExitTimestamp=Thu 2015-08-13 14:50:19 CST   # backup.service 上次执行的时间

[root@study ~]# systemctl show backup.timer
NextElapseUSecMonotonic=2d 19min 11.540653s         # 下一次执行距离 timers.target 的时间
```

如上表所示，我上次执行 backup.service 的时间是在 2015-08-13 14:50 ，由于设置两个小时执行一次，因此下次应该是 2015-08-15 14:50 执行才对！ 由于 timer 是由 timers.target 这个 unit 所管理的，而这个 timers.target 的启动时间是在 2015-08-13 14:31 ， 要注意，最终 backup.timer 所纪录的下次执行时间，其实是与 timers.target 所纪录的时间差！因此是“ 2015-08-15 14:50 - 2015-08-13 14:31 ”才对！ 所以时间差就是 2d 19min 啰！

- 一个固定日期运行的案例

上面的案例是固定周期运行一次，那如果我希望不管上面如何运行了，我都希望星期天凌晨 2 点运行这个备份程序一遍呢？请注意，因为已经存在 backup.timer 了！ 所以，这里我用 backup2.timer 来做区隔喔！

```
[root@study ~]# vim /etc/systemd/system/backup2.timer
[Unit]
Description=backup my server timer2

[Timer]
OnCalendar=Sun *-*-* 02:00:00
Persistent=true
Unit=backup.service

[Install]
WantedBy=multi-user.target

[root@study ~]# systemctl daemon-reload
[root@study ~]# systemctl enable backup2.timer
[root@study ~]# systemctl start backup2.timer
[root@study ~]# systemctl show backup2.timer
NextElapseUSecRealtime=45y 7month 1w 6d 10h 30min
```

与循环时间运行差异比较大的地方，在于这个 OnCalendar 的方法对照的时间并不是 times.target 的启动时间，而是 Unix 标准时间！ 亦即是 1970-01-01 00:00:00 去比较的！因此，当你看到最后出现的 NextElapseUSecRealtime 时，哇！下一次执行还要 45 年 + 7 个月 + 1 周 + 6 天 + 10 小时过 30 分～刚看到的时候，鸟哥确实因此揉了揉眼睛～确定没有看错... 这才了解原来比对的是“日历时间”而不是某个 unit 的启动时间啊！呵呵！

通过这样的方式，你就可以使用 systemd 的 timer 来制作属于你的时程规划服务啰！

17.5 CentOS 7.x 默认启动的服务简易说明

随着 Linux 上面软件支持性越来越多，加上自由软件蓬勃的发展，我们可以在 Linux 上面用的 daemons 真的越来越多了。所以，想要写完所有的 daemons 介绍几乎是不可能的，因此，鸟哥这里仅介绍几个很常见的 daemons 而已， 更多的信息呢，就得要麻烦你自己使用 systemctl list-unit-files --type=service 去查询啰！ 下面的建议主要是针对 Linux 单机服务器的角色来说明的，不是桌上型的环境喔！

CentOS 7.x 默认启动的服务内容	
服务名称	功能简介
abrtd	（系统）abrtd 服务可以提供使用者一些方式，让使用者可以针对不同的应用软件去设计错误登录的机制， 当软件产生问题时，使用者就可以根据 abrtd 的登录文件来进行错误克服的行为。还有其他的 abrt-xxx.service 均是使用这个服务来加强应用程序 debug 任务的。
accounts-daemon（可关闭）	（系统）使用 accountsservice 计划所提供的一系列 D-Bus 界面来进行使用者帐号信息的查询。 基本上是与 useradd, usermod, userdel 等软件有关。
alsa-X（可关闭）	（系统）开头为 alsa 的服务有不少，这些服务大部分都与音效有关！一般来说， 服务器且不开图形界面的话，这些服务可以关闭！
atd	（系统）单一的例行性工作调度，详细说明请参考第十五章。 抵挡机制的配置文件在 /etc/at.{allow,deny} 喔！
auditd	（系统）还记得前一章的 SELinux 所需服务吧？ 这就是其中一项，可以让系统需 SELinux 稽核的讯息写入 /var/log/audit/audit.log 中。
avahi-daemon（可关闭）	（系统）也是一个用户端的服务，可以通过 Zeroconf 自动的分析与管理网络。 Zeroconf 较常用在笔记本电脑与行动设备上，所以我们可以先关闭他啦！
brandbot rhel-*	（系统）这些服务大多用于开机过程中所需要的各种侦测环境的脚本，同时也提供网络界面的启动与关闭。 基本上，你不要关闭掉这些服务比较妥当！
chronyd ntpd ntpdate	（系统）都是网络校正时间的服务！一般来说，你可能需要的仅有 chronyd 而已！
cpupower	（系统）提供 CPU 的运行规范～可以参考 /etc/sysconfig/cpupower 得到更多的信息！ 这家伙与你的 CPU 使用情况有关喔！
crond	（系统）系统配置文件为 /etc/crontab，详细数据可参考第十五章的说明。
	（系统/网络）用来管理打印机的服务，可以提供网络连线的功能，有点类似打印服务器的功能哩！ 你可以在 Linux 本机上面以浏览器

cups（可关闭）	的 http://localhost:631 来管理打印机喔！由于我们目前没有打印机，所以可以暂时关闭他。
dbus	（系统）使用 D-Bus 的方式在不同的应用程序之间传送讯息， 使用的方向例如应用程序间的讯息传递、每个使用者登陆时提供的讯息数据等。
dm-event multipathd	（系统）监控设备对应表 （device mapper） 的主要服务，当然不能关掉啊！ 否则就无法让 Linux 使用我们的周边设备与储存设备了！
dmraid-activation mdmonitor	（系统）用来启动 Software RAID 的重要服务！最好不要关闭啦！虽然你可能没有 RAID。
dracut-shutdown	（系统）用来处理 initramfs 的相关行为，这与开机流程相关性较高～
ebtables	（系统/网络）通过类似 iptables 这种防火墙规则的设置方式，设计网卡作为桥接时的封包分析政策。 其实就是防火墙。不过与下面谈到的防火墙应用不太一样。如果没有使用虚拟化，或者启用了 firewalld ，这个服务可以不启动。
emergency rescue	（系统）进入紧急模式或者是救援模式的服务
firewalld	（系统/网络）就是防火墙！以前有 iptables 与 ip6tables 等防火墙机制，新的 firewalld 搭配 firewall-cmd 指令，可以快速的创建好你的防火墙系统喔！因此，从 CentOS 7.1 以后，iptables 服务的启动脚本已经被忽略了！ 请使用 firewalld 来取代 iptables 服务喔！
gdm	（系统）GNOME 的登陆管理员，就是图形界面上一个很重要的登陆管理服务！
getty@	（系统）就是要在本机系统产生几个文字界面 （tty） 登陆的服务啰！
hyper *ksm* libvirt* vmtoolsd	（系统）跟创建虚拟机有关的许多服务！如果你不玩虚拟机， 那么这些服务可以先关闭。此外，如果你的 Linux 本来就在虚拟机的环境下，那这些服务对你就没有用！因为这些服务是让实体机器来创建虚拟机的！
irqbalance	（系统）如果你的系统是多核心的硬件，那么这个服务要启动， 因为它可以自动的分配系统中断 （IRQ） 之类的硬件资源。
iscsi*	（系统）可以挂载来自网络磁盘机的服务！这个服务可以在系统内仿眞好贵的 SAN 网络磁盘。 如果你确定系统上面没有挂载这种网络磁盘，也可以将他关闭的。
kdump（可关闭）	（系统）在安装 CentOS 的章节就谈过这东西，主要是 Linux 核心如果出错时，用来纪录内存的东西。 鸟哥觉得不需要启动他！除非你是核心骇客！
lvm2-*	（系统）跟 LVM 相关性较高的许多服务，当然也不能关！不然系统上面的 LVM2 就没人管了！
microcode	（系统）Intel 的 CPU 会提供一个外挂的微指令集提供系统运行，不过，如果你没有下载 Intel 相关的指令集文件，那么这个服务不需要启动的，也不会影响系统运行。

ModemManager network NetworkManager*	（系统/网络）主要就是调制解调器、网络设置等服务！进入 CentOS 7 之后，系统似乎不太希望我们使用 network 服务了， 比较建议的是使用 NetworkManager 搭配 nmcli 指令来处理网络设置～所以，反而是 NetworkManager 要开，而 network 不用开哩！
quotaon	（系统）启动 Quota 要用到的服务喔！
rc-local	（系统）相容于 /etc/rc.d/rc.local 的调用方式！只是，你必须要让 /etc/rc.d/rc.local 具有 x 的权限后， 这个服务才能真的运行！否则，你写入 /etc/rc.d/rc.local 的脚本还是不会运行的喔！
rsyslog	（系统）这个服务可以记录系统所产生的各项讯息， 包括 /var/log/messages 内的几个重要的登录文件啊。
smartd	（系统）这个服务可以自动的侦测硬盘状态，如果硬盘发生问题的话， 还能够自动的回报给系统管理员，是个非常有帮助的服务喔！不可关闭他啊！
sysstat	（系统）事实上，我们的系统有只名为 sar 的指令会记载某些时间点下，系统的资源使用情况，包括 CPU/流量/输入输出量等， 当 sysstat 服务启动后，这些纪录的数据才能够写入到纪录档 （log） 里面去！
systemd-*	（系统）大概都是属于系统运行过程所需要的服务，没必要都不要更动它的默认状态！
plymount* upower	（系统）与图形界面的使用相关性较高的一些服务！没启动图形界面时，这些服务可以暂时不管他！

上面的服务是 CentOS 7.x 默认有启动的，这些默认启动的服务很多是针对桌面电脑所设计的，所以啰，如果你的 Linux 主机用途是在服务器上面的话，那么有很多服务是可以关闭的啦！如果你还有某些不明白的服务想要关闭的， 请务必要搞清楚该服务的功能为何喔！举例来说，那个 rsyslog 就不能关闭，如果你关掉他的话，系统就不会记录登录文件， 那你的系统所产生的警告讯息就无法记录起来，你将无法进行 debug 喔。

下面鸟哥继续说明一些可能在你的系统当中的服务，只是默认并没有启动这个服务就是了。只是说明一下， 各服务的用途还是需要您自行查询相关的文章啰。

其他服务的简易说明	
服务名称	功能简介
dovecot	（网络）可以设置 POP3/IMAP 等收受信件的服务，如果你的 Linux 主机是 email server 才需要这个服务，否则不需要启动他啦！
httpd	（网络）这个服务可以让你的 Linux 服务器成为 www server 喔！
named	（网络）这是领域名称服务器 （Domain Name System） 的服务， 这个服务非常重要，但是设置非常困难！目前应该不需要这个服务啦！
nfs nfs-server	（网络）这就是 Network Filesystem，是 Unix-Like 之间互相作为网络磁盘机的一个功能。
smb nmb	（网络）这个服务可以让 Linux 仿真成为 Windows 上面的网络上的芳邻。 如果你的 Linux 主机想要做为 Windows 用户端的网络磁盘机服务器，这玩意儿得要好好玩一玩。
vsftpd	（网络）作为文件传输服务器 （FTP） 的服务。
sshd	（网络）这个是远端连线服务器的软件功能， 这个通讯协定比 telnet 好的地方在于 sshd 在传送数据时可以进行加密喔！这个服务不要关闭他啦！
rpcbind	（网络）达成 RPC 协定的重要服务！包括 NFS, NIS 等等都需要这东西的协助！
postfix	（网络）寄件的邮件主机～因为系统还是会产生很多 email 讯息！例如 crond / atd 就会传送 email 给本机用户！ 所以这个服务千万不能关！即使你不是 mail server 也是要启用这服务才行！

17.6 重点回顾

- 早期的服务管理使用 systemV 的机制，通过 /etc/init.d/*, service, chkconfig, setup 等指令来管理服务的启动/关闭/默认启动；
- 从 CentOS 7.x 开始，采用 systemd 的机制，此机制最大功能为平行处理，并采单一指令管理 （systemctl），开机速度加快！
- systemd 将各服务定义为 unit，而 unit 又分类为 service, socket, target, path, timer 等不同的类别，方便管理与维护
- 启动/关闭/重新启动的方式为： systemctl [start|stop|restart] unit.service
- 设置默认启动/默认不启动的方式为： systemctl [enable|disable] unit.service
- 查询系统所有启动的服务用 systemctl list-units --type=service 而查询所有的服务 （含不启动） 使用 systemctl list-unit-files --type=service
- systemd 取消了以前的 runlevel 概念 （虽然还是有相容的 target），转而使用不同的 target 操作环境。常见操作环境为 multi-user.targer 与 graphical.target。 不重新开机而转不同的操作环境使用 systemctl isolate unit.target，而设置默认环境则使用 systemctl set-default unit.target
- systemctl 系统默认的配置文件主要放在 /usr/lib/systemd/system，管理员若要修改或自行设计时，则建议放在 /etc/systemd/system/ 目录下。
- 管理员应使用 man systemd.unit, man systemd.service, man systemd.timer 查询 /etc/systemd/system/ 下面配置文件的语法， 并使用 systemctl daemon-reload 载入后，才能自行撰写服务与管理服务喔！
- 除了 atd 与 crond 之外，可以 通过 systemd.timer 亦即 timers.target 的功能，来使用 systemd 的时间管理功能。
- 一些不需要的服务可以关闭喔！

17.7 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ）

- 情境仿真题：通过设置、启动、观察等机制，完整的了解一个服务的启动与观察现象。
    - 目标：了解 daemon 的控管机制，以 sshd daemon 为例；
    - 前提：需要对本章已经了解，尤其是 systemd 的管理 部分；
    - 需求：已经有 sshd 这个服务，但没有修改过端口！ 在本情境中，我们使用 sshd 这个服务来观察，主要是假设 sshd 要开立第二个服务，这个第二个服务的 port 放行于 222 ，那该如何处理？ 可以这样做看看：
    - 基本上 sshd 几乎是一定会安装的服务！只是我们还是来确认看看好了！

```
[root@study ~]# systemctl status sshd.service
sshd.service - OpenSSH server daemon
   Loaded: loaded （/usr/lib/systemd/system/sshd.service; enabled）
   Active: active （running） since Thu 2015-08-13 14:31:12 CST; 20h ago

[root@study ~]# cat /usr/lib/systemd/system/sshd.service
[Unit]
Description=OpenSSH server daemon
After=network.target sshd-keygen.service
Wants=sshd-keygen.service

[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target
```

    - 通过观察 man sshd，我们可以查询到 sshd 的配置文件位于 /etc/ssh/sshd_config 这个文件内！再 man sshd_config 也能知道原来端口是使用 Port 来规范的！ 因此，我想要创建第二个配置文件，文件名假设为 /etc/ssh/sshd2_config 这样！

```
[root@study ~]# cd /etc/ssh
[root@study ssh]# cp sshd_config sshd2_config
[root@study ssh]# vim sshd2_config
Port 222
# 随意找个地方加上这个设置值！你可以在文件的最下方加入这行也 OK 喔！
```

    - 接下来开始修改启动脚本服务档！

```
[root@study ~]# cd /etc/systemd/system
[root@study system]# cp /usr/lib/systemd/system/sshd.service sshd2.service
[root@study system]# vim sshd2.service
[Unit]
Description=OpenSSH server daemon 2
After=network.target sshd-keygen.service
Wants=sshd-keygen.service

[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -f /etc/ssh/sshd2_config -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target

[root@study system]# systemctl daemon-reload
[root@study system]# systemctl enable sshd2
[root@study system]# systemctl start sshd2
[root@study system]# tail -n 20 /var/log/messages
# semanage port -a -t PORT_TYPE -p tcp 222
    where PORT_TYPE is one of the following: ssh_port_t, vnc_port_t, xserver_por
# 认真的看！你会看到上面这两句！也就是 SELinux 的端口问题！请解决！

[root@study system]# semanage port -a -t ssh_port_t -p tcp 222
[root@study system]# systemctl start sshd2
[root@study system]# netstat -tlnp &#124; grep ssh
tcp        0      0 0.0.0.0:22     0.0.0.0:*     LISTEN      1300/sshd
tcp        0      0 0.0.0.0:222    0.0.0.0:*     LISTEN      15275/sshd
tcp6       0      0 :::22          :::*          LISTEN      1300/sshd
tcp6       0      0 :::222         :::*          LISTEN      15275/sshd
```

简答题部分：

- 使用 netstat -tul 与 netstat -tunl 有什么差异？为何会这样？使用 n 时， netstat 就不会使用主机名称与服务名称 （hostname & service*name*） 来显示， 取而代之的则是以 *IP* 及 *port number* 来显示的。*IP* 的分析与 */etc/hosts* 及 */etc/resolv.conf* 有关， 这个在未来服务器篇才会提到。至于 *port number* 则与 */etc/services* 有关，请自行参考喔！ ^^
- 你能否找出来，启动 port 3306 这个端口的服务为何？通过搜寻 /etc/services 内容，得到 port 3306 为 mysql 所启动的端口喔！查询 google， 可得到 mysql 为一种网络数据库系统软件。
- 你可以通过哪些指令查询到目前系统默认开机会启动的服务？systemctl list-units 以及 systemctl list-unit-files
- 承上，那么哪些服务“目前”是在启动的状态？结果同上！只是若要进一步的信息，应该使用 systemctl status [unit.service] 一项一项查询！

17.8 参考资料与延伸阅读

- freedesktop.org 的重要介绍：http://www.freedesktop.org/wiki/Software/systemd/
- Red Hat 官网的介绍：https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7 /html/System_Administrators_Guide/chap-Managing_Services_with_systemd.html
- man systemd.unit, man systemd.service, man systemd.kill, man systemd.timer, man systemd.time
- 关于 timer 的相关介绍：
  - archlinux.org: https://wiki.archlinux.org/index.php/Systemd/Timers
  - Janson's Blog: http://jason.the-graham.com/2013/03/06/how-to-use-systemd-timers/
  - freedesktop.org: http://www.freedesktop.org/software/systemd/man/systemd.timer.html

2002/07/10：第一次完成 2003/02/11：重新编排与加入 FAQ 2005/10/03：将原本旧版的数据移动到 此处 。 2005/10/12：经过一段时间的修订，将原本在 系统设置工具 的内容移动到此，并新增完毕！ 2009/03/25：将原本旧的基于 FC4 的数据移动到此处。 2009/04/02：加入一些默认启动的服务说明。 2009/09/14：加入情境仿真，并且修订课后练习题的部分了。 2015/08/10：将旧的基于 CentOS 5 的版本移动到这里。

第十八章、认识与分析登录文件

最近更新日期：20//

当你的 Linux 系统出现不明原因的问题时，很多人都告诉你，你要查阅一下登录文件才能够知道系统出了什么问题了，所以说， 了解登录文件是很重要的事情呢。登录文件可以记录系统在什么时间、哪个主机、哪个服务、出现了什么讯息等信息， 这些信息也包括使用者识别数据、系统故障排除须知等信息。如果你能够善用这些登录文件信息的话，你的系统出现错误时， 你将可以在第一时间发现，而且也能够从中找到解决的方案，而不是昏头转向的乱问人呢。 此外，登录文件所记录的信息量是非常大的，要人眼分析实在很困难。此时利用 shell script 或者是其他软件提供的分析工具来处理复杂的登录文件，可以帮助你很多很多喔！

18.1 什么是登录文件

“详细而确实的分析以及备份系统的登录文件”是一个系统管理员应该要进行的任务之一。 那么什么是登录文件呢？简单的说，就是记录系统活动信息的几个文件， 例如：何时、何地（来源 IP）、何人 （什么服务名称）、做了什么动作 （讯息登录啰）。 换句话说就是：记录系统在什么时候由哪个程序做了什么样的行为时，发生了何种的事件等等。

18.1.1 CentOS 7 登录文件简易说明

要知道的是，我们的 Linux 主机在背景之下有相当多的 daemons 同时在工作着，这些工作中的程序总是会显示一些讯息， 这些显示的讯息最终会被记载到登录文件当中啦。也就是说，记录这些系统的重要讯息就是登录文件的工作啦！

- 登录文件的重要性

为什么说登录文件很重要，重要到系统管理员需要随时注意他呢？我们可以这么说：

- 解决系统方面的错误：

用 Linux 这么久了，你应该偶而会发现系统可能会出现一些错误，包括硬件捉不到或者是某些系统服务无法顺利运行的情况。 此时你该如何是好？由于系统会将硬件侦测过程记录在登录文件内，你只要通过查询登录文件就能够了解系统作了啥事！ 并且由第十六章我们也知道 SELinux 与登录文件的关系更加的强烈！所以啰，查询登录文件可以克服一些系统问题啦！

- 解决网络服务的问题：

你可能在做完了某些网络服务的设置后，却一直无法顺利启动该服务，此时该怎办？去庙里面拜拜抽签吗？ 三太子大大可能无法告诉你要怎么处理呢！由于网络服务的各种问题通常都会被写入特别的登录文件， 其实你只要查询登录文件就会知道出了什么差错，还不需要请示三太子大大啦！举例来说，如果你无法启动邮件服务器 （postfix）， 那么查询一下 /var/log/maillog 通常可以得到不错的解答！

- 过往事件记录簿：

这个东西相当的重要！例如：你发现 WWW 服务 （httpd 软件） 在某个时刻流量特别大，你想要了解为什么时， 可以通过登录文件去找出该时段是哪些 IP 在连线与查询的网页数据为何，就能够知道原因。 此外，万一哪天你的系统被入侵，并且被利用来攻击他人的主机，由于被攻击主机会记录攻击者，因此你的 IP 就会被对方记录。这个时候你要如何告知对方你的主机是由于被入侵所导致的问题， 并且协助对方继续往恶意来源追查呢？呵呵！此时登录文件可是相当重要的呢！



Tips 所以我们常说“天助自助者”是真的啦！你可以通过 （1）察看屏幕上面的错误讯息与（2）登录文件的错误信息，几乎可以解决大部分的 Linux 问题！

- Linux 常见的登录文件文件名

登录文件可以帮助我们了解很多系统重要的事件，包括登陆者的部分信息，因此登录文件的权限通常是设置为仅有 root 能够读取而已。 而由于登录文件可以记载系统这么多的详细信息，所以啦，一个有经验的主机管理员会随时随地查阅一下自己的登录文件， 以随时掌握系统的最新脉动！那么常见的几个登录文件有哪些呢？一般而言，有下面几个：

- /var/log/boot.log： 开机的时候系统核心会去侦测与启动硬件，接下来开始各种核心支持的功能启动等。这些流程都会记录在 /var/log/boot.log 里面哩！ 不过这个文件只会存在这次开机启动的信息，前次开机的信息并不会被保留下来！
- /var/log/cron： 还记得第十五章例行性工作调度吧？你的 crontab 调度有没有实际被进行？ 进行过程有没有发生错误？你的 /etc/crontab 是否撰写正确？在这个登录文件内查询看看。
- /var/log/dmesg： 记录系统在开机的时候核心侦测过程所产生的各项信息。由于 CentOS 默认将开机时核心的硬件侦测过程取消显示， 因此额外将数据记录一份在这个文件中；
- /var/log/lastlog： 可以记录系统上面所有的帐号最近一次登陆系统时的相关信息。第十三章讲到的 lastlog 指令就是利用这个文件的记录信息来显示的。
- /var/log/maillog 或 /var/log/mail/*： 记录邮件的往来信息，其实主要是记录 postfix （SMTP 协定提供者） 与 dovecot （POP3 协定提供者） 所产生的讯息啦。 SMTP 是发信所使用的通讯协定， POP3 则是收信使用的通讯协定。 postfix 与 dovecot 则分别是两套达成通讯协定的软件。
- /var/log/messages： 这个文件相当的重要，几乎系统发生的错误讯息 （或者是重要的信息） 都会记录在这个文件中； 如果系统发生莫名的错误时，这个文件是一定要查阅的登录文件之一。
- /var/log/secure： 基本上，只要牵涉到“需要输入帐号密码”的软件，那么当登陆时 （不管登陆正确或错误） 都会被记录在此文件中。 包括系统的 login 程序、图形接口登陆所使用的 gdm 程序、 su, sudo 等程序、还有网络连线的 ssh, telnet 等程序， 登陆信息都会被记载在这里；

- /var/log/wtmp, /var/log/faillog： 这两个文件可以记录正确登陆系统者的帐号信息（wtmp） 与错误登陆时所使用的帐号信息 （faillog） ！ 我们在第十章谈到的 last 就是读取 wtmp 来显示的， 这对于追踪一般帐号者的使用行为很有帮助！
- /var/log/httpd/, */var/log/samba/*： 不同的网络服务会使用它们自己的登录文件来记载它们自己产生的各项讯息！上述的目录内则是个别服务所制订的登录文件。

常见的登录文件就是这几个，但是不同的 Linux distributions ，通常登录文件的文件名不会相同 （除了 /var/log/messages 之外 ）。所以说，你还是得要查阅你 Linux 主机上面的登录文件设置数据， 才能知道你的登录文件主要文件名喔！

- 登录文件所需相关服务 （daemon） 与程序

那么这些登录文件是怎么产生的呢？基本上有两种方式，一种是由软件开发商自行定义写入的登录文件与相关格式， 例如 WWW 软件 apache 就是这样处理的。另一种则是由 Linux distribution 提供的登录文件管理服务来统一管理。 你只要将讯息丢给这个服务后，他就会自己分门别类的将各种讯息放置到相关的登录文件去！CentOS 提供 rsyslog.service 这个服务来统一管理登录文件喔！

不过要注意的是，如果你任凭登录文件持续记录的话，由于系统产生的信息天天都有，那么你的登录文件的容量将会长大到无法无天～ 如果你的登录文件容量太大时，可能会导致大文件读写效率不佳的问题 （因为要从磁盘读入内存，越大的文件消耗内存量越多）。 所以啰，你需要对登录文件备份与更新。那...需要手动处理喔？当然不需要，我们可以通过 logrotate （登录文件轮替） 这玩意儿来自动化处理登录文件容量与更新的问题喔！

所谓的 logrotate 基本上，就是将旧的登录文件更改名称，然后创建一个空的登录文件，如此一来， 新的登录文件将重新开始记录，然后只要将旧的登录文件留下一阵子，嗯！那就可以达到将登录文件“轮转”的目的啦！ 此外，如果旧的记录 （大概要保存几个月吧！） 保存了一段时间没有问题，那么就可以让系统自动的将他砍掉， 免得占掉很多宝贵的硬盘空间说！

总结一下，针对登录文件所需的功能，我们需要的服务与程序有：

- systemd-journald.service：最主要的讯息收受者，由 systemd 提供的；
- rsyslog.service：主要登录系统与网络等服务的讯息；
- logrotate：主要在进行登录文件的轮替功能。

由于我们着眼点在于想要了解系统上面软件所产生的各项信息，因此本章主要针对 rsyslog.service 与 logrotate 来介绍。 接着下来我们来谈一谈怎么样规划这两个玩意儿。就由 rsyslog.service 这支程序先谈起吧！毕竟得先有登录文件，才可以进行 logrotate 呀！您说是吧！

- CentOS 7.x 使用 systemd 提供的 journalctl 日志管理

CentOS 7 除了保有既有的 rsyslog.service 之外，其实最上游还使用了 systemd 自己的登录文件日志管理功能喔！他使用的是 systemd-journald.service 这个服务来支持的。基本上，系统由 systemd 所管理，那所有经由 systemd 启动的服务，如果再启动或结束的过程中发生一些问题或者是正常的讯息， 就会将该讯息由 systemd-journald.service 以二进制的方式记录下来，之后再将这个讯息发送给 rsyslog.service 作进一步的记载。

systemd-journald.service 的记录主要都放置于内存中，因此在存取方面性能比较好～我们也能够通过 journalctl 以及 systemctl status unit.service 来查看各个不同服务的登录文件！ 这有个好处，就是登录文件可以随着个别服务让你查阅，在单一服务的处理上面，要比跑到 /var/log/messages 去大海捞针来的简易很多！ 不过，因为 system-journald.service 里面的很多观念还是沿用 rsyslog.service 相关的信息，所以，本章还是先从 rsyslog.service 先谈起，谈完之后再以 journalctl 进一步了解 systemd 是怎么去记录登录文件日志功能的呦！

18.1.2 登录文件内容的一般格式

一般来说，系统产生的讯息经过记录下来的数据中，每条讯息均会记录下面的几个重要数据：

- 事件发生的日期与时间；
- 发生此事件的主机名称；
- 启动此事件的服务名称 （如 systemd, CROND 等） 或指令与函数名称 （如 su, login..）；
- 该讯息的实际数据内容。

当然，这些信息的“详细度”是可以修改的，而且，这些信息可以作为系统除错之用呢！ 我们拿登录时一定会记载帐号信息的 /var/log/secure 为例好了：

```
[root@study ~]# cat /var/log/secure
Aug 17 18:38:06 study login: pam_unix（login:session）: session opened for user root by LC
Aug 17 18:38:06 study login: ROOT LOGIN ON tty1
Aug 17 18:38:19 study login: pam_unix（login:session）: session closed for user root
Aug 18 23:45:17 study sshd[18913]: Accepted password for dmtsai from 192.168.1.200 port 4
Aug 18 23:45:17 study sshd[18913]: pam_unix（sshd:session）: session opened for user dmtsa
Aug 18 23:50:25 study sudo: dmtsai : TTY=pts/0 ; PWD=/home/dmtsai ; USER=root ; COMMAND=/
Aug 18 23:50:25 study su: pam_unix（su-l:session）: session opened for user root by dmtsai
&#124;--日期/时间---&#124;--H--&#124;-服务与相关函数-&#124;-----------讯息说明------&gt;
```

我们拿第一笔数据 （共两行） 来说明好了，该数据是说：“在 08/17 的 18:38 左右，在名为 study 的这部主机系统上，由 login 这个程序产生的讯息，内容显示 root 在 tty1 登陆了，而相关的权限给予是通过 pam_unix 模块处理的 （共两行数据） 。”有够清楚吧！那请您自行翻译一下后面的几条讯息内容是什么喔！

其实还有很多的信息值得查阅的呢！尤其是 /var/log/messages 的内容。记得一个好的系统管理员， 要常常去“巡视”登录文件的内容喔！尤其是发生下面几种情况时：

- 当你觉得系统似乎不太正常时；

- 某个 daemon 老是无法正常启动时；
- 某个使用者老是无法登陆时；
- 某个 daemon 执行过程老是不顺畅时；

还有很多啦！反正觉得系统不太正常，就得要查询查询登录文件就是了。



Tips 提供一个鸟哥常做的检查方式。当我老是无法成功的启动某个服务时，我会在最后一次启动该服务后，立即检查登录文件， 先 （1）找到现在时间所登录的信息“第一字段”；（2）找到我想要查询的那个服务“第三字段”， （3）最后再仔细的查阅第四字段的信息，来借以找到错误点。

另外，不知道你会不会觉得很奇怪？为什么登录文件就是登录本机的数据啊～那怎么登录文件格式中，第二个字段项目是“主机名称”啊？ 这是因为登录文件可以做成登录文件服务器，可以收集来自其他服务器的登录文件数据喔！所以啰，为了了解到该讯息主要是来自于哪一部主机， 当然得要有第二个字段项目说明该信息来自哪一部主机名称啰！

18.2 rsyslog.service ：记录登录文件的服务

上一小节提到说 Linux 的登录文件主要是由 rsyslog.service 在负责，那么你的 Linux 是否有启动 rsyslog 呢？而且是否有设置开机时启动呢？呵呵！检查一下先：

```
[root@study ~]# ps aux &#124; grep rsyslog
USER  PID %CPU %MEM    VSZ   RSS TTY   STAT START  TIME COMMAND
root  750  0.0  0.1 208012  4732 ?    Ssl  Aug17  0:00 /usr/sbin/rsyslogd -n
# 瞧！确实有启动的！daemon 可执行文件名为 rsyslogd 喔！

[root@study ~]# systemctl status rsyslog.service
rsyslog.service - System Logging Service
   Loaded: loaded （/usr/lib/systemd/system/rsyslog.service; enabled）
   Active: active （running） since Mon 2015-08-17 18:37:58 CST; 2 days ago
 Main PID: 750 （rsyslogd）
   CGroup: /system.slice/rsyslog.service
           └─750 /usr/sbin/rsyslogd -n
# 也有启动这个服务，也有默认开机时也要启动这个服务！OK！正常没问题！！
```

看到 rsyslog.service 这个服务名称了吧？所以知道他已经在系统中工作啰！好了，既然本章主要是讲登录文件的服务， 那么 rsyslog.service 的配置文件在哪里？如何设置？如果你的 Linux 主机想要当作整个区网的登录文件服务器时，又该如何设置？ 下面就让我们来玩玩这玩意！

18.2.1 rsyslog.service 的配置文件：/etc/rsyslog.conf

什么？登录文件还有配置文件？喔！不是啦～是 rsyslogd 这个 daemon 的配置文件啦！ 我们现在知道 rsyslogd 可以负责主机产生的各个信息的登录，而这些信息本身是有“严重等级”之分的， 而且，这些数据最终要传送到哪个文件去是可以修改的呢，所以我们才会在一开头的地方讲说， 每个 Linux distributions 放置的登录文件文件名可能会有所差异啊！

基本上， rsyslogd 针对各种服务与讯息记录在某些文件的配置文件就是 /etc/rsyslog.conf，这个文件规定了“（1）什么服务 （2）的什么等级讯息 （3）需要被记录在哪里（设备或文件）” 这三个咚咚，所以设置的语法会是这样：

```
服务名称[.=!]讯息等级          讯息记录的文件名或设备或主机
# 下面以 mail 这个服务产生的 info 等级为例：
mail.info            /var/log/maillog_info
# 这一行说明：mail 服务产生的大于等于 info 等级的讯息，都记录到
# /var/log/maillog_info 文件中的意思。
```

我们将上面的数据简单的分为三部分来说明：

- 服务名称

rsyslogd 主要还是通过 Linux 核心提供的 syslog 相关规范来设置数据的分类的，Linux 的 syslog 本身有规范一些服务讯息， 你可以通过这些服务来储存系统的讯息。Linux 核心的 syslog 认识的服务类型主要有下面这些： （可使用 man 3 syslog 查询到相关的信息，或查询 syslog.h 这个文件来了解的！）

相对序号	服务类别	说明
0	kern（kernel）	就是核心 （kernel） 产生的讯息，大部分都是硬件侦测以及核心功能的启用
1	user	在使用者层级所产生的信息，例如后续会介绍到的用户使用 logger 指令来记录登录文件的功能
2	mail	只要与邮件收发有关的讯息记录都属于这个；
3	daemon	主要是系统的服务所产生的信息，例如 systemd 就是这个有关的讯息！
4	auth	主要与认证/授权有关的机制，例如 login, ssh, su 等需要帐号/密码的咚咚；
5	syslog	就是由 syslog 相关协定产生的信息，其实就是 rsyslogd 这支程序本身产生的信息啊！
6	lpr	亦即是打印相关的讯息啊！
7	news	与新闻群组服务器有关的东西；
8	uucp	全名为 Unix to Unix Copy Protocol，早期用于 unix 系统间的程序数据交换；
9	cron	就是例行性工作调度 cron/at 等产生讯息记录的地方；
10	authpriv	与 auth 类似，但记录较多帐号私人的信息，包括 pam 模块的运行等！
11	ftp	与 FTP 通讯协定有关的讯息输出！
16~23	local0 ~ local7	保留给本机用户使用的一些登录文件讯息，较常与终端机互动。

上面谈到的都是 Linux 核心的 syslog 函数自行制订的服务名称，软件开发商可以通过调用上述的服务名称来记录他们的软件。 举例来说， sendmail 与 postfix 及 dovecot 都是与邮件有关的软件，这些软件在设计登录文件记录时，都会主动调用 syslog 内的 mail 服务名称（LOG_MAIL）。所以上述三个软件 （sendmail, postfix, dovecot） 产生的讯息在 syslog 看起来，就会“是 mail ”类型的服务了。我们可以将这个概念绘制如下面的图示来理解：

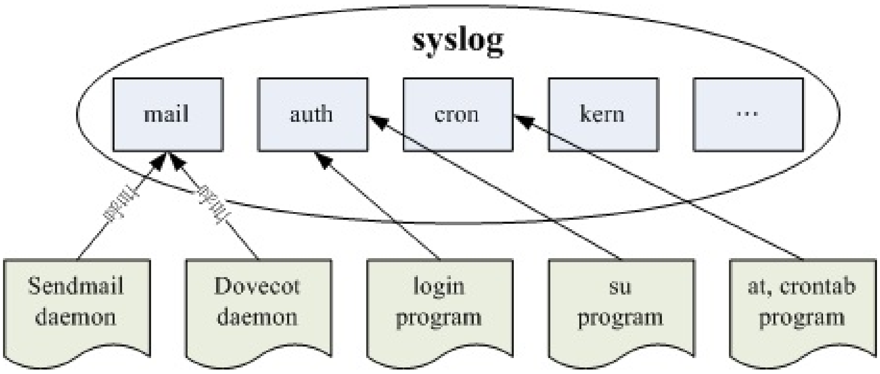


图18.2.1、syslog 所制订的服务名称与软件调用的方式

另外，每种服务所产生的数据量其实差异是很大的，举例来说， mail 的登录文件讯息多的要命， 每一封信件进入后， mail 至少需要记录“寄信人的信息；与收信者的讯息”等等； 而如果是用来做为工作站主机的，那么登陆者 （利用 login 登录主机处理事情） 的数量一定不少，那个 authpriv 所管辖的内容可就多的要命了。

为了让不同的信息放置到不同的文件当中，好让我们分门别类的进行登录文件的管理， 所以啰，将各种类别的服务之登录文件，记录在不同的文件里面，就是我们 /etc/rsyslog.conf 所要作的规范了！

- 讯息等级

同一个服务所产生的讯息也是有差别的，有启动时仅通知系统而已的一般讯息（information）， 有出现还不至于影响到正常运行的警告讯息 （warn） ，还有系统硬件发生严重错误时，所产生的重大问题讯息 （error 等等）； 讯息到底有多少种严重的等级呢？基本上，Linux 核心的 syslog 将讯息分为七个主要的等级，根据 syslog.h 的定义，讯息名称与数值的对应如下：

等级数值	等级名称	说明
7	debug	用来 debug （除错） 时产生的讯息数据；
6	info	仅是一些基本的讯息说明而已；
5	notice	虽然是正常信息，但比 info 还需要被注意到的一些信息内容；
4	warning（warn）	警示的讯息，可能有问题，但是还不至于影响到某个 daemon 运行的信息；基本上， info, notice, warn 这三个讯息都是在告知一些基本信息而已，应该还不至于造成一些系统运行困扰；
3	err（error）	一些重大的错误讯息，例如配置文件的某些设置值造成该服务服法启动的信息说明， 通常借由 err 的错误告知，应该可以了解到该服务无法启动的问题呢！
2	crit	比 error 还要严重的错误信息，这个 crit 是临界点 （critical） 的缩写，这个错误已经很严重了喔！
1	alert	警告警告，已经很有问题的等级，比 crit 还要严重！
0	emerg（panic）	疼痛等级，意指系统已经几乎要死机的状态！ 很严重的错误信息了。通常大概只有硬件出问题，导致整个核心无法顺利运行，就会出现这样的等级的讯息吧！

基本上，在 0（emerg） 到 6（info） 的等级之间，等级数值越高代表越没事，等级靠近 0 则代表事情大条了！除了 0 到 6 之外还有两个比较特殊的等级， 那就是 debug（错误侦测等级） 与 none （不需登录等级） 两个，当我们想要作一些错误侦测，或者是忽略掉某些服务的信息时， 就用这两个咚咚吧！

特别留意一下在讯息等级之前还有 [.=!] 的链接符号喔！他代表的意思是这样的：

- . ：代表“比后面还要严重的等级 （含该等级） 都被记录下来”的意思，例如： mail.info 代表只要是 mail 的信息，而且该信息等级严重于 info （含 info 本身）时，就会被记录下来的意思。
- .=：代表所需要的等级就是后面接的等级而已， 其他的不要！
- .!：代表不等于， 亦即是除了该等级外的其他等级都记录。

一般来说，我们比较常使用的是“.”这个链接符号啦！^_^

- 讯息记录的文件名或设备或主机

再来则是这个讯息要放置在哪里的设置了。通常我们使用的都是记录的文件啦！但是也可以输出到设备呦！ 例如打印机之类的！也可以记录到不同的主机上头去呢！下面就是一些常见的放置处：

- 文件的绝对路径：通常就是放在 /var/log 里头的文件啦！
- 打印机或其他：例如 /dev/lp0 这个打印机设备

- 使用者名称：显示给使用者啰！
- 远端主机：例如 @study.vbird.tsai 当然啦，要对方主机也能支持才行！
- *：代表“目前在线上的所有人”，类似 wall 这个指令的意义！

- 服务、daemon 与函数名称

看完上面的说明，相信你一定会越来越迷煳！啊！怎么会有 syslog, rsyslogd, rsyslog.service！见鬼～名称都不相同！那是啥东西？ 基本上，这几个东西你应该要这样看：

syslog	这个是 Linux 核心所提供的登录文件设计指引，所有的要求大概都写入道一个名为 syslog.h 的头文件案中。如果你想要开发与登录文件有关的软件， 那你就得要依循这个 syslog 函数的要求去设计才行！可以使用 man 3 syslog 去查询一下相关的数据！
rsyslogd	为了要达成实际上进行讯息的分类所开发的一套软件，所以，这就是最基本的 daemon 程序！
rsyslog.service	为了加入 systemd 的控制，因此 rsyslogd 的开发者设计的启动服务脚本设置！

这样简单的分类，应该比较容易了解名称上面的意义了吧？早期 CentOS 5.x 以前，要达成 syslog 的功能是由一只名为 syslogd 的 daemon 来完成的， 从 CentOS 6 以来 （包含 CentOS 7） 则是通过 rsyslogd 这个 daemon 啰！

- rsyslog.conf 语法练习

基本上，整个 rsyslog.conf 配置文件的内容参数大概就只是这样而已，下面我们来思考一些例题，好让你可以更清楚的知道如何设置 rsyslogd 啊！

例题：如果我要将我的 mail 相关的数据给他写入 /var/log/maillog 当中，那么在 /etc/rsyslog.conf 的语法如何设计？答：基本的写法是这样的：

> mail.info /var/log/maillog

注意到上面喔，当我们的等级使用 info 时，那么“任何严重于 info 等级（含 info 这个等级）之上的讯息， 都会被写入到后面接的文件之中！”这样可以了解吗？也就是说，我们可以将所有 mail 的登录信息都记录在 /var/log/maillog 里面的意思啦！

例题：我要将新闻群组数据 （news） 及例行性工作调度 （cron） 的讯息都写入到一个称为 /var/log/cronnews 的文件中，但是这两个程序的警告讯息则额外的记录在 /var/log/cronnews.warn 中， 那该如何设置我的 rsyslog.conf 呢？答：很简单啦！既然是两个程序，那么只好以分号来隔开了，此外，由于第二个指定文件中，我只要记录警告讯息， 因此设置上需要指定“.=”这个符号，所以语法成为了：

> news.;*cron.* /var/log/cronnews >
news.=warn;cron.=warn /var/log/cronnews.warn

上面那个“.=”就是在指定等级的意思啦！由于指定了等级，因此，只有这个等级的讯息才会被记录在这个文件里面呢！ 此外你也必须要注意，news 与 cron 的警告讯息也会写入 /var/log/cronnews 内喔！

例题：我的 messages 这个文件需要记录所有的信息，但是就是不想要记录 cron, mail 及 news 的信息，那么应该怎么写才好？答：可以有两种写法，分别是：

> .;news,cron,mail.none　　　　　/var/log/messages >
.;news.none;cron.none;mail.none　/var/log/messages

使用“,”分隔时，那么等级只要接在最后一个即可，如果是以“;”来分的话， 那么就需要将服务与等级都写上去啰！这样会设置了吧！

- CentOS 7.x 默认的 rsyslog.conf 内容

了解语法之后，我们来看一看 rsyslogd 有哪些系统服务已经在记录了呢？就是瞧一瞧 /etc/rsyslog.conf 这个文件的默认内容啰！ （注意！如果需要将该行做为注解时，那么就加上 # 符号就可以啦）

```
# 来自 CentOS 7.x 的相关数据
[root@study ~]# vim /etc/rsyslog.conf
#kern.*                                             /dev/console
*.info;mail.none;authpriv.none;cron.none            /var/log/messages
authpriv.*                                          /var/log/secure
mail.*                                             -/var/log/maillog
cron.*                                              /var/log/cron
*.emerg                                             :omusrmsg:*
uucp,news.crit                                      /var/log/spooler
local7.*                                            /var/log/boot.log
```

上面总共仅有 8 行设置值，每一行的意义是这样的：

1. #kern.*：只要是核心产生的讯息，全部都送到 console（终端机） 去。console 通常是由外部设备连接到系统而来， 举例来说，很多封闭型主机 （没有键盘、屏幕的系统） 可以通过连接 RS232 连接口将讯息传输到外部的系统中， 例如以笔记本电脑连接到封闭主机的 RS232 插口。这个项目通常应该是用在系统出现严重问题而无法使用默认的屏幕观察系统时， 可以通过这个项目来连接取得核心的讯息。[1]
2. *.info;mail.none;authpriv.none;cron.none：由于 mail, authpriv, cron 等类别产生的讯息较多， 且已经写入下面的数个文件中，因此在 /var/log/messages 里面就不记录这些项目。除此之外的其他讯息都写入 /var/log/messages 中。这也是为啥我们说这个 messages 文件很重要的缘故！
3. authpriv.*：认证方面的讯息均写入 /var/log/secure 文件；
4. mail.*：邮件方面的讯息则均写入 /var/log/maillog 文件；
5. cron.*：例行性工作调度均写入 /var/log/cron 文件；

6. *.emerg：当产生最严重的错误等级时，将该等级的讯息以 wall 的方式广播给所有在系统登陆的帐号得知， 要这么做的原因是希望在线的使用者能够赶紧通知系统管理员来处理这么可怕的错误问题。
7. uucp,news.crit：uucp 是早期 Unix-like 系统进行数据传递的通讯协定，后来常用在新闻群组的用途中。 news 则是新闻群组。当新闻群组方面的信息有严重错误时就写入 /var/log/spooler 文件中；
8. local7.*：将本机开机时应该显示到屏幕的讯息写入到 /var/log/boot.log 文件中；

在上面的第四行关于 mail 的记录中，在记录的文件 /var/log/maillog 前面还有个减号“ - ”是干嘛用的？由于邮件所产生的讯息比较多，因此我们希望邮件产生的讯息先储存在速度较快的内存中 （buffer） ，等到数据量够大了才一次性的将所有数据都填入磁盘内，这样将有助于登录文件的存取性能。 只不过由于讯息是暂存在内存内，因此若不正常关机导致登录信息未回填到登录文件中，可能会造成部分数据的遗失。

此外，每个 Linux distributions 的 rsyslog.conf 设置差异是颇大的，如果你想要找到相对应的登录信息时， 可得要查阅一下 /etc/rsyslog.conf 这个文件才行！否则可能会发生分析到错误的信息喔！举例来说， 鸟哥有自己写一支分析登录文件的 script，这个 script 是依据 Red Hat 系统默认的登录文件所写的，因此不同的 distributions 想要使用这支程序时，就得要自行设计与修改一下 /etc/rsyslog.conf 才行喔！否则就可能会分析到错误的信息啰。 那么如果你有自己的需要而得要修订登录文件时，该如何进行？

- 自行增加登录文件文件功能

如果你有其他的需求，所以需要特殊的文件来帮你记录时，呵呵！别客气，千万给他记录在 /etc/rsyslog.conf 当中，如此一来，你就可以重复的将许多的信息记录在不同的文件当中，以方便你的管理呢！ 让我们来作个练习题吧！如果你想要让“所有的信息”都额外写入到 /var/log/admin.log 这个文件时， 你可以怎么作呢？先自己想一想，并且作一下，再来看看下面的作法啦！

```
# 1\. 先设置好所要创建的文件设置！
[root@study ~]# vim /etc/rsyslog.conf
# Add by VBird 2015/08/19       &lt;==再次强调，自己修改的时候加入一些说明
*.info      /var/log/admin.log  &lt;==有用的是这行啦！

# 2\. 重新启动 rsyslogd 呢！
[root@study ~]# systemctl restart rsyslog.service
[root@study ~]# ll /var/log/admin.log
-rw-r--r--. 1 root root 325 Aug 20 00:54 /var/log/admin.log
# 瞧吧！创建了这个登录文件出现啰！
```

很简单吧！如此一来，所有的信息都会写入 /var/log/admin.log 里面了！

18.2.2 登录文件的安全性设置

好了，由上一个小节里面我们知道了 rsyslog.conf 的设置，也知道了登录文件内容的重要性了， 所以，如果幻想你是一个很厉害的骇客，想利用他人的计算机干坏事，然后又不想留下证据， 你会怎么作？对啦！就是离开的时候将屁股擦干净，将所有可能的讯息都给他抹煞掉， 所以第一个动脑筋的地方就是登录文件的清除工作啦～ 如果你的登录文件不见了，那该怎办？



Tips 哇！鸟哥教人家干坏事……喂！不要乱讲话～俺的意思是，如果改天你发现你的登录文件不翼而飞了， 或者是发现你的登录文件似乎不太对劲的时候，最常发现的就是网友常常会回报说，他的 /var/log 这个目录“不见了！”不要笑！这是真的事情！请记得，“赶快清查你的系统！”

伤脑筋呢！有没有办法防止登录文件被删除？或者是被 root 自己不小心变更呢？有呀！拔掉网络线或电源线就好了……呵呵！ 别担心，基本上，我们可以通过一个隐藏的属性来设置你的登录文件，成为“ 只可以增加数据，但是不能被删除 ”的状态，那么或许可以达到些许的保护！不过，如果你的 root 帐号被破解了，那么下面的设置还是无法保护的，因为你要记得“ root 是可以在系统上面进行任何事情的 ”，因此，请将你的 root 这个帐号的密码设置的安全一些！千万不要轻忽这个问题呢！



Tips 为什么登录文件还要防止被自己 （root） 不小心所修改过呢？ 鸟哥在教 Linux 的课程时，我的学生常常会举手说：“老师，我的登录文件不能记录信息了！糟糕！是不是被入侵了啊？” 怪怪！明明是计算机教室的主机，使用的是 Private IP 而且学校计中还有抵挡机制，不可能被攻击吧？ 查询了才知道原来同学很喜欢使用“ :wq ”来离开 vim 的环境，但是 rsyslogd 的登录文件只要“被编辑过”就无法继续记录！ 所以才会导致不能记录的问题。此时你得要 （1）改变使用 vim 的习惯； （2）重新启动 rsyslog.service 让他再继续提供服务才行喔！

既然如此，那么我们就来处理一下隐藏属性的东东吧！我们在第六章谈到过 lsattr 与 chattr 这两个东西啦！如果将一个文件以 chattr 设置 i 这个属性时，那么该文件连 root 都不能杀掉！而且也不能新增数据，嗯！真安全！但是，如此一来登录文件的功能岂不是也就消失了？ 因为没有办法写入呀！所以啰，我们要使用的是 a 这个属性！你的登录文件如果设置了这个属性的话，那么 他将只能被增加，而不能被删除！嗯！ 这个项目就非常的符合我们登录文件的需求啦！因此，你可以这样的增加你的登录文件的隐藏属性。



Tips 请注意，下面的这个 chattr 的设置状态："仅适合已经对 Linux 系统很有概念的朋友"来设置， 对于新手来说，建议你直接使用系统的默认值就好了，免得到最后登录文件无法写入～ 那就比较糗一点！ @_@

```
[root@study ~]# chattr +a /var/log/admin.log
[root@study ~]# lsattr /var/log/admin.log
-----a---------- /var/log/admin.log
```

加入了这个属性之后，你的 /var/log/admin.log 登录文件从此就仅能被增加，而不能被删除，直到 root 以" chattr -a /var/log/admin.log "取消这个 a 的参数之后，才能被删除或移动喔！

虽然，为了你登录文件的信息安全，这个 chattr 的 +a 旗标可以帮助你维护好这个文件， 不过，如果你的系统已经被取得 root 的权限，而既然 root 可以下达 chattr -a 来取消这个旗标，所以啰，还是有风险的啦！此外，前面也稍微提到，新手最好还是先不要增加这个旗标， 很容易由于自己的忘记，导致系统的重要讯息无法记录呢。

基本上，鸟哥认为，这个旗标最大的用处除了在保护你登录文件的数据外， 他还可以帮助你避免掉不小心写入登录文件的状况喔。要注意的是，当" 你不小心 "手动" 更动过登录文件后，例如那个 /var/log/messages ， 你不小心用 vi 打开他，离开却下达 :wq 的参数，呵呵！那么该文件未来将不会再继续进行登录动作！ "这个问题真的很常发生！由于你以 vi 储存了登录文件，则 rsyslogd 会误判为该文件已被更动过， 将导致 rsyslogd 不再写入该文件新的内容～很伤脑筋的！

要让该登录文件可以继续写入，你只要重新启动 rsyslogd.service 即可。 不过，总是比较麻烦。所以啊，如果你针对登录文件下达 chattr +a 的参数，嘿嘿！ 未来你就不需要害怕不小心更动到该文件了！因为无法写入嘛！除了可以新增之外～ ^_^

不过，也因为这个 +a 的属性让该文件无法被删除与修改，所以啰，当我们进行登录文件轮替时 （logrotate） ，将会无法移动该登录文件的文件名呢！所以会造成很大的困扰。这个困扰虽然可以使用 logrotate 的配置文件来解决，但是，还是先将登录文件的 +a 旗标拿掉吧！

```
[root@study ~]# chattr -a /var/log/admin.log
```

18.2.3 登录文件服务器的设置

我们在之前稍微提到的，在 rsyslog.conf 文件当中，可以将登录数据传送到打印机或者是远端主机上面去。这样做有什么意义呢？ 如果你将登录信息直接传送到打印机上面的话，那么万一不小心你的系统被 cracker 所入侵， 他也将你的 /var/log/ 砍掉了，怎么办？没关系啊！反

正你已经将重要数据直接以打印机记录起来了， 嘿嘿！他是无法逃开的啦！^_^

再想像一个环境，你的办公室内有十部 Linux 主机，每一部负责一个网络服务， 你为了要了解每部主机的状态，因此，你常常需要登陆这十部主机去查阅你的登录文件～ 哇！光用想的，每天要进入十部主机去查数据，想到就烦～没关系～这个时候我们可以让某一部主机当成 “登录文件服务器”，用他来记录所有的十部 linux 主机的信息，嘿嘿！这样我就直接进入一部主机就可以了！ 省时又省事，真方便～

那要怎么达到这样的功能呢？很简单啦，我们 CentOS 7.x 默认的 rsyslogd 本身就已经具有这个登录文件服务器的功能了， 只是默认并没有启动该功能而已。你可以通过 man rsyslogd 去查询一下相关的选项就能够知道啦！ 既然是登录文件服务器，那么我们的 Linux 主机当然会启动一个端口来监听了，那个默认的端口就是 UDP 或 TCP 的 port 514 喔！

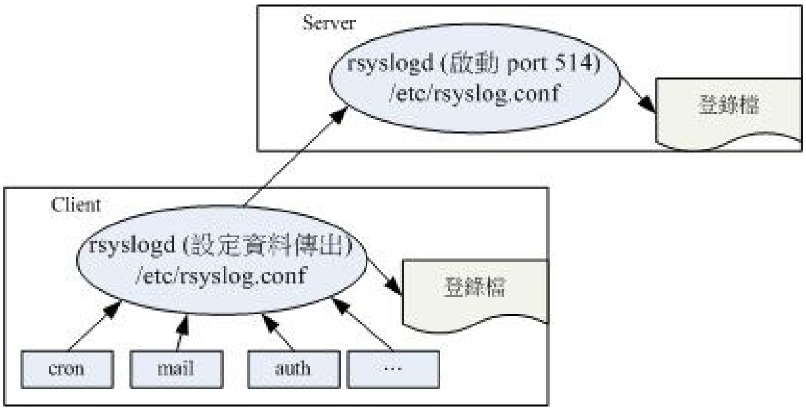


图18.2.2、登录文件服务器的架构

如上图所示，服务器会启动监听的端口，用户端则将登录文件再转出一份送到服务器去。 而既然是登录文件“服务器”，所以当然有服务器与用户端 （client） 啰！这两者的设置分别是这样的：

```
# 1\. Server 端：修改 rsyslogd 的启动配置文件，在 /etc/rsyslog.conf 内！
[root@study ~]# vim /etc/rsyslog.conf
# 找到下面这几行：
# Provides UDP syslog reception
#$ModLoad imudp
#$UDPServerRun 514

# Provides TCP syslog reception
#$ModLoad imtcp
#$InputTCPServerRun 514
# 上面的是 UDP 端口，下面的是 TCP 端口！如果你的网络状态很稳定，就用 UDP 即可。
# 不过，如果你想要让数据比较稳定传输，那么建议使用 TCP 啰！所以修改下面两行即可！
$ModLoad imtcp
$InputTCPServerRun 514

# 2\. 重新启动与观察 rsyslogd 喔！
[root@study ~]# systemctl restart rsyslog.service
[root@study ~]# netstat -ltnp &#124; grep syslog
Proto Recv-Q Send-Q Local Address   Foreign Address   State    PID/Program name
tcp        0      0 0.0.0.0:514     0.0.0.0:*         LISTEN   2145/rsyslogd
tcp6       0      0 :::514          :::*              LISTEN   2145/rsyslogd
# 嘿嘿！你的登录文件主机已经设置妥当啰！很简单吧！
```

通过这个简单的动作，你的 Linux 主机已经可以接收来自其他主机的登录信息了！ 当然啦，你必须要知道网络方面的相关基础，这里鸟哥只是先介绍，未来了解了网络相关信息后，再回头来这里瞧一瞧先！ ^_^

至于 client 端的设置就简单多了！只要指定某个信息传送到这部主机即可！ 举例来说，我们的登录文件服务器 IP 为 192.168.1.100 ，而 client 端希望所有的数据都送给主机， 所以，可以在 /etc/rsyslog.conf 里面新增这样的一行：

```
[root@study ~]# vim /etc/rsyslog.conf
*.*       @@192.168.1.100
#*.*       @192.168.1.100  # 若用 UDP 传输，设置要变这样！

[root@study ~]# systemctl restart rsyslog.service
```

再重新启动 rsyslog.service 后，立刻就搞定了！而未来主机上面的登录文件当中，每一行的“主机名称”就会显示来自不同主机的信息了。 很简单吧！ ^_^。不过你得要特别注意，使用 TCP 传输与 UDP 传输的设置不太一样！请依据你的登录文件服务器的设置值来选择你的用户端语法喔！ 接下来，让我们来谈一谈，那么如何针对登录文件来进行轮替 （rotate） 呢？

18.3 登录文件的轮替（logrotate）

假设我们已经将登录数据写入了记录文件中了，也已经利用 chattr 设置了 +a 这个属性了，那么该如何进行 logrotate 的工作呢？这里请特别留意的是："rsyslogd 利用的是 daemon 的方式来启动的， 当有需求的时候立刻就会被执行的，但是 logrotate 却是在规定的时间到了之后才来进行登录文件的轮替， 所以这个 logrotate 程序当然就是挂在 cron 下面进行的呦！" 仔细看一下 /etc/cron.daily/ 里面的文件，嘿嘿～看到了吧！ /etc/cron.daily/logrotate 就是记录了每天要进行的登录文件轮替的行为啦！ ^_^！下面我们就来谈一谈怎么样设计这个 logrotate 吧！

18.3.1 logrotate 的配置文件

既然 logrotate 主要是针对登录文件来进行轮替的动作，所以啰，他当然必须要记载" 在什么状态下才将登录文件进行轮替"的设置啊！那么 logrotate 这个程序的参数配置文件在哪里呢？呵呵！那就是：

- /etc/logrotate.conf
- /etc/logrotate.d/

那个 logrotate.conf 才是主要的参数文件，至于 logrotate.d 是一个目录， 该目录里面的所有文件都会被主动的读入 /etc/logrotate.conf 当中来进行！另外，在 /etc/logrotate.d/ 里面的文件中，如果没有规定到的一些细部设置，则以 /etc/logrotate.conf 这个文件的规定来指定为默认值！

好了，刚刚我们提到 logrotate 的主要功能就是将旧的登录文件移动成旧文件， 并且重新创建一个新的空的文件来记录，他的执行结果有点类似下面的图示：

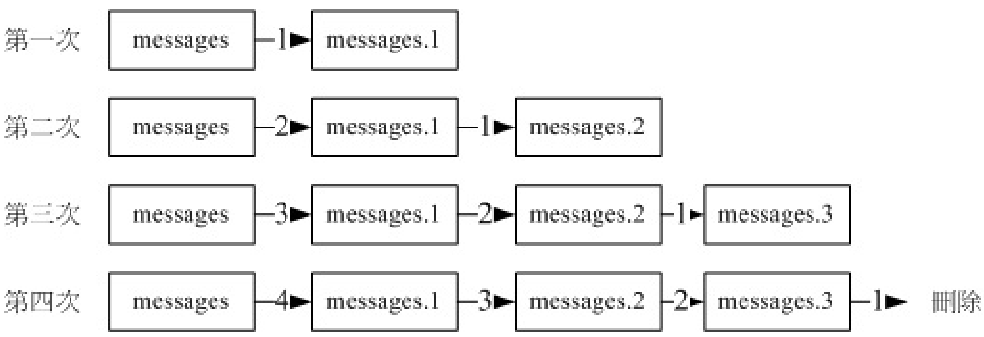


图18.3.1、登录文件进行 logrotate 的结果

由上面的图示我们可以清楚的知道，当第一次执行完 rotate 之后，原本的 messages 会变成 messages.1 而且会制造一个空的 messages 给系统来储存登录文件。而第二次执行之后，则 messages.1 会变成 messages.2 而 messages 会变成 messages.1 ，又造成一个空的

messages 来储存登录文件！那么如果我们仅设置保留三个登录文件而已的话，那么执行第四次时，则 messages.3 这个文件就会被删除，并由后面的较新的保存登录文件所取代！基本的工作就是这样啦！

不过近年来磁盘空间容量比较大了，加上管理员又担心登录文件数据真的给它不见去，因此，你可能已经发现到，最近的登录文件轮替后的文件名已经会加上日期参数， 然后源源不绝的保留在你的系统上耶～虽然这个设置是可以修订的，不过，鸟哥也真的希望保留日期的文件名延伸记录，真的比较不用担心未来要找问题时， 登录文件却已经 GG 了...

那么多久进行一次这样的 logrotate 工作呢？这些都记录在 logrotate.conf 里面，我们来看一下默认的 logrotate 的内容吧！

```
[root@study ~]# vim /etc/logrotate.conf
# 下面的设置是 "logrotate 的默认设置值" ，如果个别的文件设置了其他的参数，
# 则将以个别的文件设置为主，若该文件没有设置到的参数则以这个文件的内容为默认值！

weekly    &lt;==默认每个礼拜对登录文件进行一次 rotate 的工作
rotate 4  &lt;==保留几个登录文件呢？默认是保留四个！
create    &lt;==由于登录文件被更名，因此创建一个新的来继续储存之意！
dateext   &lt;==就是这个设置值！可以让被轮替的文件名称加上日期作为文件名喔！
#compress &lt;==被更动的登录文件是否需要压缩？如果登录文件太大则可考虑此参数启动

include /etc/logrotate.d
# 将 /etc/logrotate.d/ 这个目录中的所有文件都读进来执行 rotate 的工作！

/var/log/wtmp {       &lt;==仅针对 /var/log/wtmp 所设置的参数
    monthly           &lt;==每个月一次，取代每周！
    create 0664 root utmp &lt;==指定新建文件的权限与所属帐号/群组
    minsize 1M        &lt;==文件大小一定要超过 1M 后才进行 rotate （略过时间参数）
    rotate 1          &lt;==仅保留一个，亦即仅有 wtmp.1 保留而已。
}
# 这个 wtmp 可记录登陆者与系统重新开机时的时间与来源主机及登陆期间的时间。
# 由于具有 minsize 的参数，因此不见得每个月一定会进行一次喔！要看文件大小。
# 由于仅保留一个登录文件而已，不满意的话可以将他改成 rotate 5 吧！
```

由这个文件的设置我们可以知道 /etc/logrotate.d 其实就是由 /etc/logrotate.conf 所规划出来的目录，所以，其实我们可以将所有的数据都给他写入 /etc/logrotate.conf 即可，但是这样一来这个文件就实在是太复杂了，尤其是当我们使用很多的服务在系统上面时， 每个服务都要去修改 /etc/logrotate.conf 的设置也似乎不太合理～ 所以，如果独立出来一个目录，那么每个以 RPM 打包方式所创建的服务的登录文件轮替设置， 就可以独自成为一个文件，并且放置到 /etc/logrotate.d/ 当中即可，真是方便又合理的做法啊！ ^_^

一般来说，这个 /etc/logrotate.conf 是“默认的轮替状态”而已， 我们的各个服务都可以拥有自己的登录文件轮替设置，你也可以自行修改成自己喜欢的样式啊！ 例如，如果你的系统的空间够大，并且担心除错以及骇客的问题，那么可以：

- 将 rotate 4 改成 rotate 9 左右，以保存较多的备份文件。不过如果已经加上 dateext 的参数，那这个项目就不用更动了！
- 大部分的登录文件不需要 compress 啰！但是空间太小就需要 compress ！尤其是很占硬盘空间的 httpd 更需要 compress 的！

好了，上面我们大致介绍了 /var/log/wtmp 这个文件的设置，现在你知道了 logrotate.conf 的设置语法是：

```
登录文件的绝对路径文件名 ... {
    个别的参数设置值，如 monthly, compress 等等
}
```

下面我们再以 /etc/logrotate.d/syslog 这个轮替 rsyslog.service 服务的文件，来看看该如何设置他的 rotate 呢？

```
[root@study ~]# vim /etc/logrotate.d/syslog
/var/log/cron
/var/log/maillog
/var/log/messages
/var/log/secure
/var/log/spooler
{
    sharedscripts
    postrotate
        /bin/kill -HUP `cat /var/run/syslogd.pid 2&gt; /dev/null` 2&gt; /dev/null &#124;&
    endscript
}
```

在上面的语法当中，我们知道正确的 logrotate 的写法为：

- 文件名：被处理的登录文件绝对路径文件名写在前面，可以使用空白字符分隔多个登录文件；
- 参数：上述文件名进行轮替的参数使用 { } 包括起来；
- 执行脚本：可调用外部指令来进行额外的命令下达，这个设置需与 sharedscripts .... endscript 设置合用才行。至于可用的环境为：
    - prerotate：在启动 logrotate 之前进行的指令，例如修改登录文件的属性等动作；
    - postrotate：在做完 logrotate 之后启动的指令，例如重新启动 （kill -HUP） 某个服务！
    - Prerotate 与 postrotate 对于已加上特殊属性的文件处理上面，是相当重要的执行程序！

那么 /etc/logrotate.d/syslog 内设置的 5 个文件的轮替功能就变成了：

- 该设置只对 /var/log/ 内的 cron, maillog, messages, secure, spooler 有效；
- 登录文件轮替每周一次、保留四个、且轮替下来的登录文件不进行压缩（未更改默认值）；
- 轮替完毕后 （postrotate） 取得 syslog 的 PID 后，以 kill -HUP 重新启动 syslogd

假设我们有针对 /var/log/messages 这个文件增加 chattr +a 的属性时， 依据 logrotate 的工作原理，我们知道，这个 /var/log/messages 将会被更名成为 /var/log/messages.1 才是。但是由于加上这个 +a 的参数啊，所以更名是不可能成功的！ 那怎么办呢？呵呵！就利用

prerotate 与 postrotate 来进行登录文件轮替前、后所需要作的动作啊！ 果真如此时，那么你可以这样修改一下这个文件喔！

```
[root@study ~]# vim /etc/logrotate.d/syslog
/var/log/cron
/var/log/maillog
/var/log/messages
/var/log/secure
/var/log/spooler
{
    sharedscripts
    prerotate
       /usr/bin/chattr -a /var/log/messages
    endscript
    sharedscripts
    postrotate
        /bin/kill -HUP `cat /var/run/syslogd.pid 2&gt; /dev/null` 2&gt; /dev/null &#124;&
        /usr/bin/chattr +a /var/log/messages
    endscript
}
```

看到否？就是先给他去掉 a 这个属性，让登录文件 /var/log/messages 可以进行轮替的动作，然后执行了轮替之后，再给他加入这个属性！请特别留意的是，那个 /bin/kill -HUP ... 的意义，这一行的目的在于将系统的 rsyslogd 重新以其参数文件 （rsyslog.conf） 的数据读入一次！也可以想成是 reload 的意思啦！ 由于我们创建了一个新的空的记录文件，如果不执行此一行来重新启动服务的话， 那么记录的时候将会发生错误呦！（请回到第十六章读一下 kill 后面的 signal 的内容说明）

18.3.2 实际测试 logrotate 的动作

好了，设置完成之后，我们来测试看看这样的设置是否可行呢？给他执行下面的指令：

```
[root@study ~]# logrotate [-vf] logfile
选项与参数：
-v  ：启动显示模式，会显示 logrotate 运行的过程喔！
-f  ：不论是否符合配置文件的数据，强制每个登录文件都进行 rotate 的动作！

范例一：执行一次 logrotate 看看整个流程为何？
[root@study ~]# logrotate -v /etc/logrotate.conf
reading config file /etc/logrotate.conf &lt;==读取主要配置文件
including /etc/logrotate.d              &lt;==调用外部的设置
reading config file chrony              &lt;==就是外部设置啊！
....（中间省略）....
Handling 18 logs                        &lt;==共有 18 个登录文件被记录
....（中间省略）....
rotating pattern: /var/log/cron
/var/log/maillog
/var/log/messages
/var/log/secure
/var/log/spooler
 weekly （52 rotations）
empty log files are not rotated, old logs are removed
considering log /var/log/cron
  log does not need rotating
considering log /var/log/maillog
  log does not need rotating
considering log /var/log/messages       &lt;==开始处理 messages
  log does not need rotating            &lt;==因为时间未到，不需要更动！
....（下面省略）....

范例二：强制进行 logrotate 的动作
[root@study ~]# logrotate -vf /etc/logrotate.conf
....（前面省略）....
rotating log /var/log/messages, log-&gt;rotateCount is 52
dateext suffix '-20150820'
glob pattern '-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
compressing log with: /bin/gzip
....（下面省略）....
# 看到否？整个 rotate 的动作就是这样一步一步进行的～

[root@study ~]# ll /var/log/messages*; lsattr /var/log/messages
-rw-------. 1 root root    143 Aug 20 01:45 /var/log/messages
-rw-------. 1 root root 167125 Aug 20 01:40 /var/log/messages-20150820
-----a---------- /var/log/messages  &lt;==主动加入 a 的隐藏属性啰！
```

上面那个 -f 具有“强制执行”的意思，如果一切的设置都没有问题的话，那么理论上，你的 /var/log 这个目录就会起变化啰！而且应该不会出现错误讯息才对！嘿嘿！这样就 OK 了！很棒不是吗？！

由于 logrotate 的工作已经加入 crontab 里头了！所以现在每天系统都会自动的给他查看 logrotate 啰！不用担心的啦！只是要注意一下那个 /var/log/messages 里头是否常常有类似下面的字眼：

> Aug 20 01:45:34 study rsyslogd: [origin software="rsyslogd" swVersion="7.4.7" x-pid="2145" x-info="http://www.rsyslog.com"] rsyslogd was HUPed

这说明的是 rsyslogd 重新启动的时间啦 （就是因为 /etc/logrotate.d/syslog 的设置之缘故！） 下面我们来进行一些例题的练习，让你更详细的了解 logrotate 的功用啊！

18.3.3 自订登录文件的轮替功能

假设前提是这样的，前一小节当中，假设你已经创建了 /var/log/admin.log 这个文件， 现在，你想要将该文件加上 +a 这个隐藏标签，而且设置下面的相关信息：

- 登录文件轮替一个月进行一次；
- 该登录文件若大于 10MB 时，则主动进行轮替，不需要考虑一个月的期限；
- 保存五个备份文件；
- 备份文件需要压缩

那你可以怎么样设置呢？呵呵～很简单啊！看看下面的动作吧！

```
# 1\. 先创建 +a 这个属性啊！
[root@study ~]# chattr +a /var/log/admin.log
[root@study ~]# lsattr /var/log/admin.log
-----a---------- /var/log/admin.log
[root@study ~]# mv /var/log/admin.log /var/log/admin.log.1
mv: cannot move `/var/log/admin.log' to `/var/log/admin.log.1': Operation not permitted
# 这里确定了加入 a 的隐藏属性！所以 root 无法移动此登录文件！

# 2\. 开始创建 logrotate 的配置文件，增加一个文件在 /etc/logrotate.d 内就对了！
[root@study ~]# vim /etc/logrotate.d/admin
# This configuration is from VBird 2015/08/19
/var/log/admin.log {
        monthly   &lt;==每个月进行一次
        size=10M  &lt;==文件大小大于 10M 则开始处置
        rotate 5  &lt;==保留五个！
        compress  &lt;==进行压缩工作！
        sharedscripts
        prerotate
                /usr/bin/chattr -a /var/log/admin.log
        endscript
        sharedscripts
        postrotate
                /bin/kill -HUP `cat /var/run/syslogd.pid 2&gt; /dev/null` 2&gt; /dev/null
                /usr/bin/chattr +a /var/log/admin.log
        endscript
}

# 3\. 测试一下 logrotate 相关功能的信息显示：
[root@study ~]# logrotate -v /etc/logrotate.conf
....（前面省略）....
rotating pattern: /var/log/admin.log  10485760 Bytes （5 rotations）
empty log files are rotated, old logs are removed
considering log /var/log/admin.log
  log does not need rotating
not running prerotate script, since no logs will be rotated
not running postrotate script, since no logs were rotated
....（下面省略）....
# 因为还不足一个月，文件也没有大于 10M，所以不需进行轮替！

# 4\. 测试一下强制 logrotate 与相关功能的信息显示：
[root@study ~]# logrotate -vf /etc/logrotate.d/admin
reading config file /etc/logrotate.d/admin
reading config file /etc/logrotate.d/admin

Handling 1 logs

rotating pattern: /var/log/admin.log  forced from command line （5 rotations）
empty log files are rotated, old logs are removed
considering log /var/log/admin.log
  log needs rotating
rotating log /var/log/admin.log, log-&gt;rotateCount is 5
dateext suffix '-20150820'
glob pattern '-[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'
renaming /var/log/admin.log.5.gz to /var/log/admin.log.6.gz （rotatecount 5, logstart 1,
old log /var/log/admin.log.5.gz does not exist
renaming /var/log/admin.log.4.gz to /var/log/admin.log.5.gz （rotatecount 5, logstart 1,
```

```
old log /var/log/admin.log.4.gz does not exist
renaming /var/log/admin.log.3.gz to /var/log/admin.log.4.gz （rotatecount 5, logstart 1,
old log /var/log/admin.log.3.gz does not exist
renaming /var/log/admin.log.2.gz to /var/log/admin.log.3.gz （rotatecount 5, logstart 1,
old log /var/log/admin.log.2.gz does not exist
renaming /var/log/admin.log.1.gz to /var/log/admin.log.2.gz （rotatecount 5, logstart 1,
old log /var/log/admin.log.1.gz does not exist
renaming /var/log/admin.log.0.gz to /var/log/admin.log.1.gz （rotatecount 5, logstart 1,
old log /var/log/admin.log.0.gz does not exist
log /var/log/admin.log.6.gz doesn't exist -- won't try to dispose of it
running prerotate script
fscreate context set to system_u:object_r:var_log_t:s0
renaming /var/log/admin.log to /var/log/admin.log.1
running postrotate script
compressing log with: /bin/gzip

[root@study ~]# lsattr /var/log/admin.log*
-----a---------- /var/log/admin.log
---------------- /var/log/admin.log.1.gz  &lt;==有压缩过喔！
```

看到了吗？通过这个方式，我们可以创建起属于自己的 logrotate 设置文件， 很简便吧！尤其是要注意的， /etc/rsyslog.conf 与 /etc/logrotate.d/* 文件常常要搭配起来，例如刚刚我们提到的两个案例中所创建的 /var/log/admin.log 就是一个很好的例子～创建后，还要使用 logrotate 来轮替啊！ ^_^

18.4 systemd-journald.service 简介

过去只有 rsyslogd 的年代中，由于 rsyslogd 必须要开机完成并且执行了 rsyslogd 这个 daemon 之后，登录文件才会开始记录。所以，核心还得要自己产生一个 klogd 的服务， 才能将系统在开机过程、启动服务的过程中的信息记录下来，然后等 rsyslogd 启动后才传送给它来处理～

现在有了 systemd 之后，由于这玩意儿是核心唤醒的，然后又是第一支执行的软件，它可以主动调用 systemd-journald 来协助记载登录文件～ 因此在开机过程中的所有信息，包括启动服务与服务若启动失败的情况等等，都可以直接被记录到 systemd-journald 里头去！

不过 systemd-journald 由于是使用于内存的登录文件记录方式，因此重新开机过后，开机前的登录文件信息当然就不会被记载了。 为此，我们还是建议启动 rsyslogd 来协助分类记录！也就是说， systemd-journald 用来管理与查询这次开机后的登录信息，而 rsyslogd 可以用来记录以前及现在的所以数据到磁盘文件中，方便未来进行查询喔！



Tips 虽然 systemd-journald 所记录的数据其实是在内存中，但是系统还是利用文件的型态将它记录到 /run/log/ 下面！ 不过我们从前面几章也知道， /run 在 CentOS 7 其实是内存内的数据，所以重新开机过后，这个 /run/log 下面的数据当然就被刷新，旧的当然就不再存在了！

18.4.1 使用 journalctl 观察登录信息

那么 systemd-journald.service 的数据要如何叫出来查阅呢？很简单！就通过 journalctl 即可！让我们来瞧瞧这个指令可以做些什么事？

```
[root@study ~]# journalctl [-nrpf] [--since TIME] [--until TIME] _optional
选项与参数：
默认会秀出全部的 log 内容，从旧的输出到最新的讯息
-n  ：秀出最近的几行的意思～找最新的信息相当有用
-r  ：反向输出，从最新的输出到最旧的数据
-p  ：秀出后面所接的讯息重要性排序！请参考前一小节的 rsyslogd 信息
-f  ：类似 tail -f 的功能，持续显示 journal 日志的内容（实时监测时相当有帮助！）
--since --until：设置开始与结束的时间，让在该期间的数据输出而已
_SYSTEMD_UNIT=unit.service ：只输出 unit.service 的信息而已
_COMM=bash ：只输出与 bash 有关的信息
_PID=pid   ：只输出 PID 号码的信息
_UID=uid   ：只输出 UID 为 uid 的信息
SYSLOG_FACILITY=[0-23] ：使用 syslog.h 规范的服务相对序号来调用出正确的数据！

范例一：秀出目前系统中所有的 journal 日志数据
[root@study ~]# journalctl
-- Logs begin at Mon 2015-08-17 18:37:52 CST, end at Wed 2015-08-19 00:01:01 CST. --
Aug 17 18:37:52 study.centos.vbird systemd-journal[105]: Runtime journal is using 8.0M （r
 142.4M, leaving 213.6M of free 1.3G, current limit 142.4M）.
Aug 17 18:37:52 study.centos.vbird systemd-journal[105]: Runtime journal is using 8.0M （r
 142.4M, leaving 213.6M of free 1.3G, current limit 142.4M）.
Aug 17 18:37:52 study.centos.vbird kernel: Initializing cgroup subsys cpuset
Aug 17 18:37:52 study.centos.vbird kernel: Initializing cgroup subsys cpu
.....（中间省略）.....
Aug 19 00:01:01 study.centos.vbird run-parts（/etc/cron.hourly）[19268]: finished 0anacron
Aug 19 00:01:01 study.centos.vbird run-parts（/etc/cron.hourly）[19270]: starting 0yum-hou
Aug 19 00:01:01 study.centos.vbird run-parts（/etc/cron.hourly）[19274]: finished 0yum-hou
# 从这次开机以来的所有数据都会显示出来！通过 less 一页页翻动给管理员查阅！数据量相当大！

范例二：（1）仅显示出 2015/08/18 整天以及（2）仅今天及（3）仅昨天的日志数据内容
[root@study ~]# journalctl --since "2015-08-18 00:00:00" --until "2015-08-19 00:00:00"
[root@study ~]# journalctl --since today
[root@study ~]# journalctl --since yesterday --until today

范例三：只找出 crond.service 的数据，同时只列出最新的 10 笔即可
[root@study ~]# journalctl _SYSTEMD_UNIT=crond.service -n 10

范例四：找出 su, login 执行的登录文件，同时只列出最新的 10 笔即可
[root@study ~]# journalctl _COMM=su _COMM=login -n 10

范例五：找出讯息严重等级为错误 （error） 的讯息！
[root@study ~]# journalctl -p err

范例六：找出跟登录服务 （auth, authpriv） 有关的登录文件讯息
[root@study ~]# journalctl SYSLOG_FACILITY=4 SYSLOG_FACILITY=10
# 更多关于 syslog_facility 的数据，请参考 18.2.1 小节的内容啰！
```

基本上，有 journalctl 就真的可以搞定你的讯息数据啰！全部的数据都在这里面耶～再来假设一下，你想要了解到登录文件的实时变化， 那又该如何处置呢？现在，请开两个终端机，让我们来处理处理！

```
# 第一号终端机，请使用下面的方式持续侦测系统！
[root@study ~]# journalctl -f
# 这时系统会好像卡住～其实不是卡住啦！是类似 tail -f 在持续的显示登录文件信息的！

# 第二号终端机，使用下面的方式随便发一封 email 给系统上的帐号！
[root@study ~]# echo "testing" &#124; mail -s 'tset' dmtsai
# 这时，你会发现到第一号终端机竟然一直输出一些讯息吧！没错！这就对了！
```

如果你有一些必须要侦测的行为，可以使用这种方式来实时了解到系统出现的讯息～而取消 journalctl -f 的方法，就是 [crtl]+c 啊！

18.4.2 logger 指令的应用

上面谈到的是叫出登录文件给我们查阅，那换个角度想，“如果你想要让你的数据储存到登录文件当中”呢？那该如何是好？ 这时就得要使用 logger 这个好用的家伙了！这个家伙可以传输很多信息，不过，我们只使用最简单的本机信息传递～ 更多的用法就请您自行 man logger 啰！

```
[root@study ~]# logger [-p 服务名称.等级] "讯息"
选项与参数：
服务名称.等级 ：这个项目请参考 rsyslogd 的本章后续小节的介绍；

范例一：指定一下，让 dmtsai 使用 logger 来传送数据到登录文件内
[root@study ~]# logger -p user.info "I will check logger command"
[root@study ~]# journalctl SYSLOG_FACILITY=1 -n 3
-- Logs begin at Mon 2015-08-17 18:37:52 CST, end at Wed 2015-08-19 18:03:17 CST. --
Aug 19 18:01:01 study.centos.vbird run-parts（/etc/cron.hourly）[29710]: starting 0yum-hou
Aug 19 18:01:01 study.centos.vbird run-parts（/etc/cron.hourly）[29714]: finished 0yum-hou
Aug 19 18:03:17 study.centos.vbird dmtsai[29753]: I will check logger command
```

现在，让我们来瞧一瞧，如果我们之前写的 backup.service 服务中，如果使用手动的方式来备份，亦即是使用 "/backups/backup.sh log" 来执行备份时， 那么就通过 logger 来记录备份的开始与结束的时间！该如何是好呢？这样作看看！

```
[root@study ~]# vim /backups/backup.sh
#!/bin/bash

if [ "${1}" == "log" ]; then
        logger -p syslog.info "backup.sh is starting"
fi
source="/etc /home /root /var/lib /var/spool/{cron,at,mail}"
target="/backups/backup-system-$（date +%Y-%m-%d）.tar.gz"
[ ! -d /backups ] && mkdir /backups
tar -zcvf ${target} ${source} &&gt; /backups/backup.log
if [ "${1}" == "log" ]; then
        logger -p syslog.info "backup.sh is finished"
fi

[root@study ~]# /backups/backup.sh log
[root@study ~]# journalctl SYSLOG_FACILITY=5 -n 3
Aug 19 18:09:37 study.centos.vbird dmtsai[29850]: backup.sh is starting
Aug 19 18:09:54 study.centos.vbird dmtsai[29855]: backup.sh is finished
```

通过这个玩意儿，我们也能够将数据自行处置到登录文件当中啰！

18.4.3 保存 journal 的方式

再强调一次，这个 systemd-journald.servicd 的讯息是不会放到下一次开机后的，所以，重新开机后，那之前的记录通通会遗失。 虽然我们大概都有启动 rsyslogd 这个服务来进行后续的登录文件放置，不过如果你比较喜欢 journalctl 的存取方式，那么可以将这些数据储存下来喔！

基本上，systemd-journald.service 的配置文件主要参考 /etc/systemd/journald.conf 的内容，详细的参数你可以参考 man 5 journald.conf 的数据。 因为默认的情况下面，配置文件的内容应该已经符合我们的需求，所以这边鸟哥就不再修改配置文件了。只是如果想要保存你的 journalctl 所读取的登录文件， 那么就得要创建一个 /var/log/journal 的目录，并且处理一下该目录的权限，那么未来重新启动 systemd-journald.service 之后， 日志登录文件就会主动的复制一份到 /var/log/journal 目录下啰！

```
# 1\. 先处理所需要的目录与相关权限设置
[root@study ~]# mkdir /var/log/journal
[root@study ~]# chown root:systemd-journal /var/log/journal
[root@study ~]# chmod 2775 /var/log/journal

# 2\. 重新启动 systemd-journald 并且观察备份的日志数据！
[root@study ~]# systemctl restart systemd-journald.service
[root@study ~]# ll /var/log/journal/
drwxr-sr-x. 2 root systemd-journal 27 Aug 20 02:37 309eb890d09f440681f596543d95ec7a
```

你得要注意的是，因为现在整个日志登录文件的容量会持续长大，因此你最好还是观察一下你系统能用的总容量喔！避免不小心文件系统的容量被灌爆！ 此外，未来在 /run/log 下面就没有相关的日志可以观察了！因为移动到 /var/log/journal 下面来啰！

其实鸟哥是这样想的，既然我们还有 rsyslog.service 以及 logrotate 的存在，因此这个 systemd-journald.service 产生的登录文件， 个人建议最好还是放置到 /run/log 的内存当中，以加快存取的速度！而既然 rsyslog.service 可以存放我们的登录文件， 似乎也没有必要再保存一份 journal 登录文件到系统当中就是了。单纯的建议！如何处理，依照您的需求即可喔！

18.5 分析登录文件

登录文件的分析是很重要的！你可以自行以 vim 或者是 journalctl 进入登录文件去查阅相关的信息。而系统也提供一些软件可以让你从登录文件中取得数据， 例如之前谈过的 last, lastlog, dmesg 等等指令。不过，这些数据毕竟都非常的分散，如果你想要一口气读取所有的登录信息， 其实有点困扰的。不过，好在 CentOS 有提供 logwatch 这个登录文件分析程序，你可以借由该程序来了解登录文件信息。 此外，鸟哥也依据 Red Hat 系统的 journalctl 搭配 syslog 函数写了一支小程序给大家使用喔！

18.5.1 CentOS 默认提供的 logwatch

虽然有一些有用的系统指令，不过，要了解系统的状态，还是得要分析整个登录文件才行～ 事实上，目前已经有相当多的登录文件分析工具，例如 CentOS 7.x 上面默认的 logwatch 这个套件所提供的分析工具， 他会每天分析一次登录文件，并且将数据以 email 的格式寄送给 root 呢！ 你也可以直接到 logwatch 的官方网站上面看看：

- http://www.logwatch.org/

不过在我们的安装方式里面，默认并没有安装 logwatch 就是了！所以，我们先来安装一下 logwatch 这套软件再说。假设你已经将 CentOS 7.1 的原版光盘挂载在 /mnt 当中了，那使用下面的方式来处理即可：

```
[root@study ~]# yum install /mnt/Packages/perl-5.*.rpm
&gt;  /mnt/Packages/perl-Date-Manip-*.rpm \
&gt;  /mnt/Packages/perl-Sys-CPU-*.rpm \
&gt;  /mnt/Packages/perl-Sys-MemInfo-*.rpm \
&gt;  /mnt/Packages/logwatch-*.rpm
# 得要安装数个软件才能够顺利的安装好 logwatch 喔！当然，如果你有网络，直接安装就好了！

[root@study ~]# ll /etc/cron.daily/0logwatch
-rwxr-xr-x. 1 root root 434 Jun 10  2014 /etc/cron.daily/0logwatch

[root@study ~]# /etc/cron.daily/0logwatch
```

安装完毕以后，logwatch 就已经写入 cron 的运行当中了！详细的执行方式你可以参考上表中 0logwatch 文件内容来处理，未来则每天会送出一封 email 给 root 查阅就是了。因为我们刚刚安装，那可以来分析一下吗？很简单啦！你就直接执行 0logwatch 即可啊！如上表最后一个指令的示意。因为鸟哥的测试机目前的服务很少， 所以产生的信息量也不多，因此执行的速度很快。比较忙的系统信息量比较大，分析过程会花去一小段时间。如果顺利执行完毕， 那请用 root 的身份去读一下 email 啰！

```
[root@study ~]# mail
Heirloom Mail version 12.5 7/5/10\.  Type ? for help.
"/var/spool/mail/root": 5 messages 2 new 4 unread
&gt;N  4 root                  Thu Jul 30 19:35  29/763   "testing at job"
 N  5 logwatch@study.cento  Thu Aug 20 17:55  97/3045  "Logwatch for study.centos.vbird
& 5
```

```
Message  5:
From root@study.centos.vbird  Thu Aug 20 17:55:23 2015
Return-Path: &lt;root@study.centos.vbird&gt;
X-Original-To: root
Delivered-To: root@study.centos.vbird
To: root@study.centos.vbird
From: logwatch@study.centos.vbird
Subject: Logwatch for study.centos.vbird （Linux）
Auto-Submitted: auto-generated
Precedence: bulk
Content-Type: text/plain; charset="iso-8859-1"
Date: Thu, 20 Aug 2015 17:55:23 +0800 （CST）
Status: R

# logwatch 会先说明分析的时间与 logwatch 版本等等信息
 ################### Logwatch 7.4.0 （03/01/11） ####################
        Processing Initiated: Thu Aug 20 17:55:23 2015
        Date Range Processed: yesterday
                              （ 2015-Aug-19 ）
                              Period is day.
        Detail Level of Output: 0
        Type of Output/Format: mail / text
        Logfiles for Host: study.centos.vbird
 ##################################################################

# 开始一项一项的数据进行分析！分析得很有道理啊！
 --------------------- pam_unix Begin ------------------------
 su-l:
    Sessions Opened:
       dmtsai -&gt; root: 2 Time（s）
 ---------------------- pam_unix End ------------------------

 --------------------- Postfix Begin ------------------------
      894   Bytes accepted                                     894
      894   Bytes delivered                                    894
 ========   ================================================
        2   Accepted                                       100.00%
 --------   ------------------------------------------------
        2   Total                                          100.00%
 ========   ================================================
        2   Removed from queue
        2   Delivered
 ---------------------- Postfix End ------------------------

 --------------------- SSHD Begin ------------------------
 Users logging in through sshd:
    dmtsai:
       192.168.1.200: 2 times
 Received disconnect:
    11: disconnected by user : 1 Time（s）
 ---------------------- SSHD End ------------------------

 --------------------- Sudo （secure-log） Begin ------------------------
 dmtsai =&gt; root
 --------------
 /bin/su                        -   2 Time（s）.
 ---------------------- Sudo （secure-log） End ------------------------

# 当然也得说明一下目前系统的磁盘使用状态喔！
 --------------------- Disk Space Begin ------------------------
 Filesystem                 Size  Used Avail Use% Mounted on
 /dev/mapper/centos-root     10G  3.7G  6.3G  37% /
 devtmpfs                   1.4G     0  1.4G   0% /dev
 /dev/vda2                 1014M  141M  874M  14% /boot
 /dev/vda4                 1014M   33M  982M   4% /srv/myproject
 /dev/mapper/centos-home    5.0G  642M  4.4G  13% /home
 /dev/mapper/raidvg-raidlv  1.5G   33M  1.5G   3% /srv/raidlvm
 ---------------------- Disk Space End ------------------------
```

由于鸟哥的测试用主机尚未启动许多服务，所以分析的项目很少。若你的系统已经启动许多服务的话， 那么分析的项目理应会多很多才对。

18.5.2 鸟哥自己写的登录文件分析工具：

虽然已经有了类似 logwatch 的工具，但是鸟哥自己想要分析的数据毕竟与对方不同～所以啰，鸟哥就自己写了一支小程序 （shell script 的语法） 用来分析自己的登录文件，这支程序分析的登录文件主要由 journalctl 所产生，而且只会抓前一天的登录文件来分析而已～ 若比对 rsyslog.service 所产生的登录文件，则主要用到下面几个对应的文件名 （虽然真的没用到！ ^_^） ：

- /var/log/secure
- /var/log/messages
- /var/log/maillog

当然啦，还不只这些啦，包括各个主要常见的服务，如 pop3, mail, ftp, su 等会使用到 pam 的服务， 都可以通过鸟哥写的这个小程序来分析与处理呢～整个数据还会输出一些系统信息。如果你想要使用这个程序的话， 欢迎下载：

- http://linux.vbird.org//linux_basic/0570syslog//logfile_centos7.tar.gz

安装的方法也很简单，你只要将上述的文件在根目录下面解压缩，自然就会将 cron 调度与相对应的文件放到正确的目录去。 基本上鸟哥会用到的目录有 /etc/cron.d 以及 /root/bin/logfile 而已！鸟哥已经写了一个 crontab 在文件中，设置每日 00:10 去分析一次系统登录文件。 不过请注意，这次鸟哥使用的登录文件真的是来自于 journalctl ，所以 CentOS 6 以前的版本千万不要使用喔！现在假设我将下载的文件放在跟目录，所以：

```
[root@study ~]# tar -zxvf /logfile_centos7.tar.gz -C /
[root@study ~]# cat /etc/cron.d/vbirdlogfile
10 0 * * * root /bin/bash /root/bin/logfile/logfile.sh &&gt; /dev/null

[root@study ~]# sh /root/bin/logfile/logfile.sh
# 开始尝试分析系统的登录文件，依据你的登录文件大小，分析的时间不固定！

[root@study ~]# mail
# 自己找到刚刚输出的结果，该结果的输出有点像下面这样：
Heirloom Mail version 12.5 7/5/10\.  Type ? for help.
"/var/spool/mail/root": 9 messages 4 new 7 unread
 N  8 root                  Thu Aug 20 19:26  60/2653  "study.centos.vbird logfile analys
&gt;N  9 root                  Thu Aug 20 19:37  59/2612  "study.centos.vbird logfile ana
& 9

# 先看看你的硬件与操作系统的相关情况，尤其是 partition 的使用量更需要随时注意！
=============== system summary =================================
Linux kernel  :  Linux version 3.10.0-229.el7.x86_64 （builder@kbuilder.dev.centos.org）
CPU informatin:  2 Intel（R） Xeon（R） CPU E5-2650 v3 @ 2.30GHz
CPU speed     :  2299.996 MHz
hostname is   :  study.centos.vbird
Network IP    :  192.168.1.100
Check time    :  2015/August/20 19:37:25 （ Thursday ）
Summary date  :  Aug 20
Up times      :  3 days, 59 min,
Filesystem summary:
       Filesystem               Type      Size  Used Avail Use% Mounted on
       /dev/mapper/centos-root  xfs        10G  3.7G  6.3G  37% /
       devtmpfs                 devtmpfs  1.4G     0  1.4G   0% /dev
       tmpfs                    tmpfs     1.4G   48K  1.4G   1% /dev/shm
       tmpfs                    tmpfs     1.4G  8.7M  1.4G   1% /run
       tmpfs                    tmpfs     1.4G     0  1.4G   0% /sys/fs/cgroup
       /dev/vda2                xfs      1014M  141M  874M  14% /boot
       /dev/vda4                xfs      1014M   33M  982M   4% /srv/myproject
       /dev/mapper/centos-home  xfs       5.0G  642M  4.4G  13% /home
       /dev/mapper/raidvg-raidlv xfs      1.5G   33M  1.5G   3% /srv/raidlvm
       /dev/sr0                 iso9660   7.1G  7.1G     0 100% /mnt

# 这个程序会将针对 internet 与内部监听的端口分开来显示！
================= Ports 的相关分析信息 =======================
主机启用的 port 与相关的 process owner：
对外部接口开放的 ports （PID&#124;owner&#124;command）
       tcp 21&#124;（root）&#124;/usr/sbin/vsftpd /etc/vsftpd/vsftpd.conf
       tcp 22&#124;（root）&#124;/usr/sbin/sshd -D
       tcp 25&#124;（root）&#124;/usr/libexec/postfix/master -w
       tcp 222&#124;（root）&#124;/usr/sbin/sshd -f /etc/ssh/sshd2_config -D
       tcp 514&#124;（root）&#124;/usr/sbin/rsyslogd -n
       tcp 555&#124;（root）&#124;/usr/sbin/vsftpd /etc/vsftpd/vsftpd2.conf

# 以下针对有启动的服务个别进行分析！
================= SSH 的登录文件信息汇整 =======================
今日没有使用 SSH 的纪录

================= Postfix 的登录文件信息汇整 ===================
使用者信箱受信次数：
```

目前鸟哥都是通过这支程序去分析自己管理的主机，然后再据以了解系统状况，如果有特殊状况则实时进行系统处理！ 而且鸟哥都是将上述的 email 调整成自己可以在 Internet 上面读到的邮件，这样我每天都可以收到正确的登录文件分析信息哩！

18.6 重点回顾

- 登录文件可以记录一个事件的何时、何地、何人、何事等四大信息，故系统有问题时务必查询登录文件；
- 系统的登录文件默认都集中放置到 /var/log/ 目录内，其中又以 messages 记录的信息最多！
- 登录文件记录的主要服务与程序为： systemd-journald.service, rsyslog.service, rsyslogd
- rsyslogd 的配置文件在 /etc/rsyslog.conf ，内容语法为："服务名称.等级 记载设备或文件"
- 通过 linux 的 syslog 函数查询，了解上述服务名称有 kernel, user, mail...从 0 到 23 的服务序号
- 承上，等级从不严重到严重依序有 info, notice, warning, error, critical, alert, emergency 等
- rsyslogd 本身有提供登录文件服务器的功能，通过修改 /etc/rsyslog.conf 内容即可达成；
- logrotate 程序利用 crontab 来进行登录文件的轮替功能；
- logrotate 的配置文件为 /etc/logrotate.conf ，而额外的设置则可写入 /etc/logrotate.d/* 内；
- 新的 CentOS 7 由于内置 systemd-journald.service 的功能，可以使用 journalctl 直接从内存读出登录文件，查询性能较佳
- logwatch 为 CentOS 7 默认提供的一个登录文件分析软件。

18.7 本章习题

（要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看） 实作题：

- 请在你的 CentOS 7.x 上面，依照鸟哥提供的 logfile.sh 去安装，并将结果取出分析看看。

简答题部分：

- 如果你想要将 auth 这个服务的结果中，只要讯息等级高于 warn 就给予发送 email 到 root 的信箱，该如何处理？利用 vim 去编辑 /etc/rsyslog.conf 文件，内容为 auth.warn root
- 启动系统登录信息时，需要启动哪两个 daemon 呢？systemd-journald.service, rsyslog.service
- rsyslogd 以及 logrotate 个别通过什么机制来执行？rsyslogd 为 stand alone daemon 的机制；logrotate 则是通过 crontab 来执行的！只是个指令而已。

18.8 参考资料与延伸阅读

- [1]关于 console 的说明可以参考下面的链接： http://en.wikipedia.org/wiki/Console http://publib.boulder.ibm.com/infocenter/systems/index.jsp?topic=/com.ibm.aix.files/doc/aixfiles/console.htm
- 关于 logfile 也有网友提供英文版喔： http://phorum.vbird.org/viewtopic.php?f=10&t=34996&p=148198

2002/06/24：第一次完成 2003/02/11：重新编排与加入 FAQ 2005/10/12：旧的文章已经被移动到 此处。 2005/10/24：终于写完了～啊！怎么写这么久？？ 2006/07/23：修改了 /etc/logrotate.d/syslog 的设置数据 2009/03/31：将旧的基于 FC4 版本的数据移动至 此处 2009/09/14：加入了一些例题而已。这一篇太简单了～想不到什么好的题目说～ 2010/12/24：感谢网友 eujiang 提供的英文版 logfile.sh 程序喔！ 2015/08/14：将旧的基于 CentOS 5 的版本移动到 这里，有需要的前往观察！

第十九章、开机流程、模块管理与 Loader

最近更新日期：20//

系统开机其实是一项非常复杂的程序，因为核心得要侦测硬件并载入适当的驱动程序后， 接下来则必须要调用程序来准备好系统运行的环境，以让使用者能够顺利的操作整部主机系统。 如果你能够理解开机的原理，那么将有助于你在系统出问题时能够很快速的修复系统喔！ 而且还能够顺利的配置多重操作系统的多重开机问题。为了多重开机的问题，你就不能不学学 grub2 这个 Linux 下面优秀的开机管理程序 （boot loader）。 而在系统运行期间，你也得要学会管理核心模块呢！

19.1 Linux 的开机流程分析

如果想要多重开机，那要怎么安装系统？如果你的 root 密码忘记了，那要如何救援？如果你的默认登陆模式为图形界面，那要如何在开机时直接指定进入纯文本模式？ 如果你因为 /etc/fstab 设置错误，导致无法顺利挂载根目录，那要如何在不重灌的情况下修订你的 /etc/fstab 让它变成正常？这些都需要了解开机流程， 那你说，这东西重不重要啊？

19.1.1 开机流程一览

既然开机是很严肃的一件事，那我们就来了解一下整个开机的过程吧！好让大家比较容易发现开机过程里面可能会发生问题的地方，以及出现问题后的解决之道！ 不过，由于开机的过程中，那个开机管理程序 （Boot Loader） 使用的软件可能不一样，例如目前各大 Linux distributions 的主流为 grub2，但早期 Linux 默认是使用 grub1 或 LILO ，台湾地区则很多朋友喜欢使用 spfdisk 。 但无论如何，我们总是得要了解整个 boot loader 的工作情况，才能了解为何进行多重开机的设置时， 老是听人家讲要先安装 Windows 再安装 Linux 的原因～

假设以个人计算机架设的 Linux 主机为例 （先回到第零章计算机概论看看相关的硬件常识喔）， 当你按下电源按键后计算机硬件会主动的读取 BIOS 或 UEFI BIOS 来载入硬件信息及进行硬件系统的自我测试， 之后系统会主动的去读取第一个可开机的设备 （由 BIOS 设置的） ，此时就可以读入开机管理程序了。

开机管理程序可以指定使用哪个核心文件来开机，并实际载入核心到内存当中解压缩与执行， 此时核心就能够开始在内存内活动，并侦测所有硬件信息与载入适当的驱动程序来使整部主机开始运行， 等到核心侦测硬件与载入驱动程序完毕后，一个最阳春的操作系统就开始在你的 PC 上面跑了。

主机系统开始运行后，此时 Linux 才会调用外部程序开始准备软件执行的环境，并且实际的载入所有系统运行所需要的软件程序哩！ 最后系统就会开始等待你的登陆与操作啦！简单来说，系统开机的经过可以汇整成下面的流程的：

1. 载入 BIOS 的硬件信息与进行自我测试，并依据设置取得第一个可开机的设备；
2. 读取并执行第一个开机设备内 MBR 的 boot Loader （亦即是 grub2, spfdisk 等程序）；
3. 依据 boot loader 的设置载入 Kernel ，Kernel 会开始侦测硬件与载入驱动程序；
4. 在硬件驱动成功后，Kernel 会主动调用 systemd 程序，并以 default.target 流程开机；
    - systemd 执行 sysinit.target 初始化系统及 basic.target 准备操作系统；
    - systemd 启动 multi-user.target 下的本机与服务器服务；
    - systemd 执行 multi-user.target 下的 /etc/rc.d/rc.local 文件；
    - systemd 执行 multi-user.target 下的 getty.target 及登陆服务；
    - systemd 执行 graphical 需要的服务

大概的流程就是上面写的那个样子啦，你会发现 systemd 这个家伙占的比重非常重！ 所以我们才会在第十六章的 pstree 指令中谈到这家伙。 那每一个程序的内容主要是在干嘛呢？下面就分别来谈一谈吧！

19.1.2 BIOS, boot loader 与 kernel 载入

我们在第二章曾经谈过简单的开机流程与 MBR 的功能，以及大容量磁盘需要使用的 GPT 分区表格式等。 详细的数据请再次回到第二章好好的阅读一下，我们这里为了讲解方便起见，将后续会用到的专有名词先做个综合解释：

- BIOS：不论传统 BIOS 还是 UEFI BIOS 都会被简称为 BIOS；
- MBR：虽然分区表有传统 MBR 以及新式 GPT，不过 GPT 也有保留一块相容 MBR 的区块，因此，下面的说明在安装 boot loader 的部份， 鸟哥还是简称为 MBR 喔！总之，MBR 就代表该磁盘的最前面可安装 boot loader 的那个区块就对了！

- BIOS, 开机自我测试与 MBR/GPT

我们在第零章的计算机概论就曾谈过计算机主机架构， 在个人计算机架构下，你想要启动整部系统首先就得要让系统去载入 BIOS （Basic Input Output System），并通过 BIOS 程序去载入 CMOS 的信息，并且借由 CMOS 内的设置值取得主机的各项硬件设置， 例如 CPU 与周边设备的沟通频率啊、开机设备的搜寻顺序啊、硬盘的大小与类型啊、 系统时间啊、各周边总线的是否启动 Plug and Play （PnP, 随插即用设备） 啊、 各周边设备的 I/O 位址啊、以及与 CPU 沟通的 IRQ 岔断等等的信息。

在取得这些信息后，BIOS 还会进行开机自我测试 （Power-on Self Test, POST） [1]。 然后开始执行硬件侦测的初始化，并设置 PnP 设备，之后再定义出可开机的设备顺序，接下来就会开始进行开机设备的数据读取了。

由于我们的系统软件大多放置到硬盘中嘛！所以 BIOS 会指定开机的设备好让我们可以读取磁盘中的操作系统核心文件。 但由于不同的操作系统他的文件系统格式不相同，因此我们必须要以一个开机管理程序来处理核心文件载入 （load） 的问题， 因此这个开机管理程序就被称为 Boot Loader 了。那这个 Boot Loader 程序安装在哪里呢？就在开机设备的第一个扇区 （sector） 内，也就是我们一直谈到的 MBR （Master Boot Record, 主要开机记录区）。

那你会不会觉得很奇怪啊？既然核心文件需要 loader 来读取，那每个操作系统的 loader 都不相同， 这样的话 BIOS 又是如何读取 MBR 内的 loader 呢？很有趣的问题吧！其实 BIOS 是通过硬件的 INT 13 中断功能来读取 MBR 的，也就是说，只要 BIOS 能够侦测的到你的磁盘 （不论该磁盘是 SATA 还是 SAS 接口），那他就有办法通过 INT 13 这条信道来读取该磁盘的第一个扇区内的 MBR 软件啦！[2]这样 boot loader 也就能够被执行啰！



Tips 我们知道每颗硬盘的最前面区块含有 MBR 或 GPT 分区表的提供 loader 的区块，那么如果我的主机上面有两颗硬盘的话， 系统会去哪颗硬盘的最前面区块读取 boot loader 呢？这个就得要看 BIOS 的设置了。 基本上，我们常常讲的“系统的 MBR”其实指的是 第一个开机设备的 **MBR** 才对！ 所以，改天如果你要将开机管理程序安装到某颗硬盘的 MBR 时， 要特别注意当时系统的“第一个开机设备”是哪个，否则会安装到错误的硬盘上面的 MBR 喔！重要重要！

- Boot Loader 的功能

刚刚说到 Loader 的最主要功能是要认识操作系统的文件格式并据以载入核心到内存中去执行。 由于不同操作系统的文件格式不一致，因此每种操作系统都有自己的 boot loader 啦！用自己的 loader 才有办法载入核心文件嘛！那问题就来啦，你应该有听说过多重操作系统吧？也就是在一部主机上面安装多种不同的操作系统。 既然你 （1）必须要使用自己的 loader 才能够载入属于自己的操作系统核心，而 （2）系统的 MBR 只有一个，那你怎么会有办法同时在一部主机上面安装 Windows 与 Linux 呢？

这就得要回到第七章的磁盘文件系统去回忆一下文件系统功能了。 其实每个文件系统 （filesystem, 或者是 partition） 都会保留一块开机扇区 （boot sector） 提供操作系统安装 boot loader ， 而通常操作系统默认都会安装一份 loader 到他根目录所在的文件系统的 boot sector 上。如果我们在一部主机上面安装 Windows 与 Linux 后，该 boot sector, boot loader 与 MBR 的相关性会有点像下图：

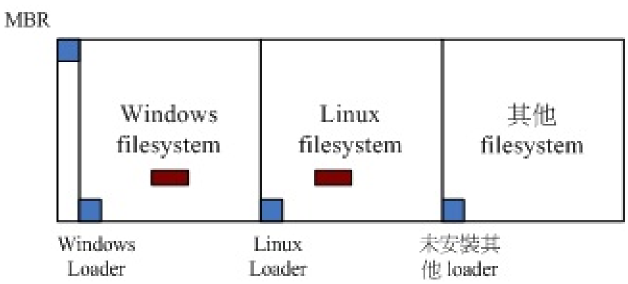


图19.1.1、boot loader 安装在 MBR, boot sector 与操作系统的关系

如上图所示，每个操作系统默认是会安装一套 boot loader 到他自己的文件系统中 （就是每个 filesystem 左下角的方框） ，而在 Linux 系统安装时，你可以选择将 boot loader 安装到 MBR 去，也可以选择不安装。 如果选择安装到 MBR 的话，那理论上你在 MBR 与 boot sector 都会保有一份 boot loader 程序的。 至于 Windows 安装时，他默认会主动的将 MBR 与 boot sector 都装上一份 boot loader！所以啦， 你会发现安装多重操作系统时，你的 MBR 常常会被不同的操作系统的 boot loader 所覆盖啦！ ^_^

我们刚刚提到的两个问题还是没有解决啊！虽然各个操作系统都可以安装一份 boot loader 到他们的 boot sector 中， 这样操作系统可以通过自己的 boot loader 来载入核心了。问题是系统的 MBR 只有一个哩！ 你要怎么执行 boot sector 里面的 loader 啊？这个我们得要回忆一下第二章约略提过的 boot loader 的功能了。boot loader 主要的功能如下：

- 提供菜单：使用者可以选择不同的开机项目，这也是多重开机的重要功能！
- 载入核心文件：直接指向可开机的程序区段来开始操作系统；
- 转交其他 **loader**：将开机管理功能转交给其他 loader 负责。

由于具有菜单功能，因此我们可以选择不同的核心来开机。而由于具有控制权转交的功能，因此我们可以载入其他 boot sector 内的 loader 啦！不过 Windows 的 loader 默认不具有控制权转交的功能，因此你不能使用 Windows 的 loader 来载入 Linux 的 loader 喔！这也是为啥第二章谈到 MBR 与多重开机时，会特别强调先装 Windows 再装 Linux 的缘故。 我们将上述的三个功能以下面的图示来解释你就看的懂了！（与第二章的图示也非常类似啦！）

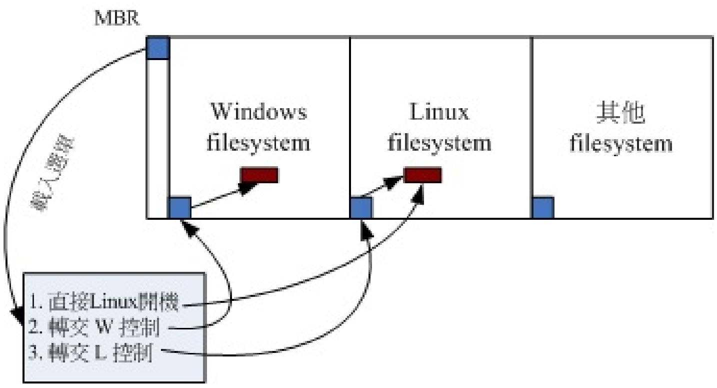


图19.1.2、开机管理程序的菜单功能与控制权转交功能示意图

如上图所示，我的 MBR 使用 Linux 的 grub2 这个开机管理程序，并且里面假设已经有了三个菜单， 第一个菜单可以直接指向 Linux 的核心文件并且直接载入核心来开机；第二个菜单可以将开机管理程控权交给 Windows 来管理，此时 Windows 的 loader 会接管开机流程，这个时候他就能够启动 windows 了。第三个菜单则是使用 Linux 在 boot sector 内的开机管理程序，此时就会跳出另一个 grub2 的菜单啦！了解了吗？

- 菜单一：MBR（grub2） --> kernel file --> booting
- 菜单二：MBR（grub2） --> boot sector（Windows loader） --> Windows kernel --> booting
- 菜单三：MBR（grub2） --> boot sector（grub2） --> kernel file --> booting

而最终 boot loader 的功能就是“载入 kernel 文件”啦！

- 载入核心侦测硬件与 initramfs 的功能

当我们借由 boot loader 的管理而开始读取核心文件后，接下来， Linux 就会将核心解压缩到内存当中， 并且利用核心的功能，开始测试与驱动各个周边设备，包括储存设备、CPU、网卡、声卡等等。 此时 Linux 核心会以自己的功能重新侦测一次硬件，而不一定会使用 BIOS

侦测到的硬件信息喔！也就是说，核心此时才开始接管 BIOS 后的工作了。 那么核心文件在哪里啊？一般来说，他会被放置到 /boot 里面，并且取名为 /boot/vmlinuz 才对！

```
[root@study ~]# ls --format=single-column -F /boot
config-3.10.0-229.el7.x86_64                &lt;==此版本核心被编译时选择的功能与模块配置文件
grub/                                       &lt;==旧版 grub1 ，不需要理会这目录了！
grub2/                                      &lt;==就是开机管理程序 grub2 相关数据目录
initramfs-0-rescue-309eb890d3d95ec7a.img    &lt;==下面几个为虚拟文件系统文件！这一个是用来救援的
initramfs-3.10.0-229.el7.x86_64.img         &lt;==正常开机会用到的虚拟文件系统
initramfs-3.10.0-229.el7.x86_64kdump.img    &lt;==核心出问题时会用到的虚拟文件系统
System.map-3.10.0-229.el7.x86_64            &lt;==核心功能放置到内存位址的对应表
vmlinuz-0-rescue-309eb890d09543d95ec7a*     &lt;==救援用的核心文件
vmlinuz-3.10.0-229.el7.x86_64*              &lt;==就是核心文件啦！最重要者！
```

从上表中的特殊字体，我们也可以知道 CentOs 7.x 的 Linux 核心为 3.10.0-229.el7.x86_64 这个版本！为了硬件开发商与其他核心功能开发者的便利， 因此 Linux 核心是可以通过动态载入核心模块的 （就请想成驱动程序即可），这些核心模块就放置在 /lib/modules/ 目录内。 由于模块放置到磁盘根目录内 （要记得 /lib 不可以与 / 分别放在不同的 partition ！）， 因此在开机的过程中核心必须要挂载根目录，这样才能够读取核心模块提供载入驱动程序的功能。 而且为了担心影响到磁盘内的文件系统，因此开机过程中根目录是以只读的方式来挂载的喔。

一般来说，非必要的功能且可以编译成为模块的核心功能，目前的 Linux distributions 都会将他编译成为模块。 因此 USB, SATA, SCSI... 等磁盘设备的驱动程序通常都是以模块的方式来存在的。 现在来思考一种情况，假设你的 linux 是安装在 SATA 磁盘上面的，你可以通过 BIOS 的 INT 13 取得 boot loader 与 kernel 文件来开机，然后 kernel 会开始接管系统并且侦测硬件及尝试挂载根目录来取得额外的驱动程序。

问题是，核心根本不认识 SATA 磁盘，所以需要载入 SATA 磁盘的驱动程序， 否则根本就无法挂载根目录。但是 SATA 的驱动程序在 /lib/modules 内，你根本无法挂载根目录又怎么读取到 /lib/modules/ 内的驱动程序？是吧！非常的两难吧！在这个情况之下，你的 Linux 是无法顺利开机的！ 那怎办？没关系，我们可以通过虚拟文件系统来处理这个问题。

虚拟文件系统 （Initial RAM Disk 或 Initial RAM Filesystem） 一般使用的文件名为 /boot/initrd 或 /boot/initramfs ，这个文件的特色是，他也能够通过 boot loader 来载入到内存中，然后这个文件会被解压缩并且在内存当中仿真成一个根目录， 且此仿真在内存当中的文件系统能够提供一支可执行的程序，通过该程序来载入开机过程中所最需要的核心模块， 通常这些模块就是 USB, RAID, LVM, SCSI 等文件系统与磁盘接口的驱动程序啦！等载入完成后， 会帮助核心重新调用 systemd 来开始后续的正常开机流程。

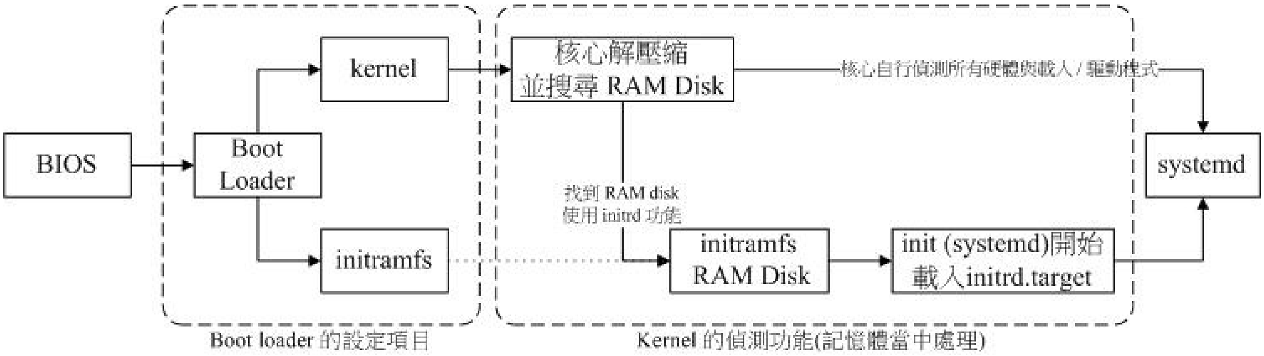


图19.1.3、BIOS 与 boot loader 及核心载入流程示意图

如上图所示，boot loader 可以载入 kernel 与 initramfs ，然后在内存中让 initramfs 解压缩成为根目录， kernel 就能够借此载入适当的驱动程序，最终释放虚拟文件系统，并挂载实际的根目录文件系统，就能够开始后续的正常开机流程。 更详细的 initramfs 说明，你可以自行使用 man initrd 去查阅看看。 下面让我们来了解一下 CentOS 7.x 的 initramfs 文件内容有什么吧！ ^_^

```
# 1\. 先来直接看一下 initramfs 里面的内容有些啥数据？
[root@study ~]# lsinitrd /boot/initramfs-3.10.0-229.el7.x86_64.img
# 首先会调用出 initramfs 最前面文件开始的许多数据介绍，这部份会占用一些容量！
Image: /boot/initramfs-3.10.0-229.el7.x86_64.img: 18M
========================================================================
Early CPIO image
========================================================================
drwxr-xr-x   3 root     root            0 May  4 17:56 .
-rw-r--r--   1 root     root            2 May  4 17:56 early_cpio
drwxr-xr-x   3 root     root            0 May  4 17:56 kernel
drwxr-xr-x   3 root     root            0 May  4 17:56 kernel/x86
drwxr-xr-x   2 root     root            0 May  4 17:56 kernel/x86/microcode
-rw-r--r--   1 root     root        10240 May  4 17:56 kernel/x86/microcode/GenuineIntel.
========================================================================
Version: dracut-033-240.el7

Arguments: -f

dracut modules:  # 开始一堆模块的载入行为
bash
nss-softokn
.....（中间省略）.....
========================================================================
drwxr-xr-x  12 root     root            0 May  4 17:56 .
crw-r--r--   1 root     root       5,   1 May  4 17:56 dev/console
crw-r--r--   1 root     root       1,  11 May  4 17:56 dev/kmsg
crw-r--r--   1 root     root       1,   3 May  4 17:56 dev/null
.....（中间省略）.....
lrwxrwxrwx   1 root     root           23 May  4 17:56 init -&gt; usr/lib/systemd/systemd
.....（中间省略）.....
drwxr-xr-x   2 root     root            0 May  4 17:56 var/lib/lldpad
lrwxrwxrwx   1 root     root           11 May  4 17:56 var/lock -&gt; ../run/lock
lrwxrwxrwx   1 root     root           10 May  4 17:56 var/log -&gt; ../run/log
lrwxrwxrwx   1 root     root            6 May  4 17:56 var/run -&gt; ../run
========================================================================
# 最后则会列出这个 initramfs 里头的所有文件！也就是说，这个 initramfs 文件大概存着两部份，
# 先是文件开始宣告的许多文件部份，再来才是真的会被核心取用的全部附加的文件数据！
```

从上面我们大概知道了这个 initramfs 里头含有两大区块，一个是事先宣告的一些数据，包括 kernel/x86/microcode/GenuineIntel.bin 这些东西。 在这些数据后面，才是真的我们的核心会去读取的重要文件～如果看一下文件的内容，你会发现到 init 那只程序已经被 systemd 所取

代啰！这样理解否？ 好～如果你想要进一步将这个文件解开的话，那得要先将前面的 kernel/x86/microcode/GenuineIntel.bin 之前的文件先去除掉，这样才能够顺利的解开。 因此，得要这样进行：

```
# 1\. 先将 /boot 下面的文件进行去除前面不需要的文件开始数据部份。
[root@study ~]# mkdir /tmp/initramfs
[root@study ~]# cd /tmp/initramfs
[root@study initramfs]# dd if=/boot/initramfs-3.10.0-229.el7.x86_64.img of=initramfs.gz \
&gt;  bs=11264 skip=1
[root@study initramfs]# ll initramfs.gz; file initramfs.gz
-rw-r--r--. 1 root root 18558166 Aug 24 19:38 initramfs.gz
initramfs.gz: gzip compressed data, from Unix, last modified: Mon May  4 17:56:47 2015,
 max compression

# 2\. 从上面看到文件是 gzip 压缩文件，所以将它解压缩后，再查阅一下文件的类型！
[root@study initramfs]# gzip -d initramfs.gz
[root@study initramfs]# file initramfs
initramfs: ASCII cpio archive （SVR4 with no CRC）

# 3\. 解开后又产生一个 cpio 文件，得要将它用 cpio 的方法解开！加上不要绝对路径的参数较保险！
[root@study initramfs]# cpio -i -d -H newc --no-absolute-filenames &lt; initramfs
[root@study initramfs]# ll
lrwxrwxrwx.  1 root root        7 Aug 24 19:40 bin -&gt; usr/bin
drwxr-xr-x.  2 root root       42 Aug 24 19:40 dev
drwxr-xr-x. 12 root root     4096 Aug 24 19:40 etc
lrwxrwxrwx.  1 root root       23 Aug 24 19:40 init -&gt; usr/lib/systemd/systemd
-rw-r--r--.  1 root root 42263552 Aug 24 19:38 initramfs
lrwxrwxrwx.  1 root root        7 Aug 24 19:40 lib -&gt; usr/lib
lrwxrwxrwx.  1 root root        9 Aug 24 19:40 lib64 -&gt; usr/lib64
drwxr-xr-x.  2 root root        6 Aug 24 19:40 proc
drwxr-xr-x.  2 root root        6 Aug 24 19:40 root
drwxr-xr-x.  2 root root        6 Aug 24 19:40 run
lrwxrwxrwx.  1 root root        8 Aug 24 19:40 sbin -&gt; usr/sbin
-rwxr-xr-x.  1 root root     3041 Aug 24 19:40 shutdown
drwxr-xr-x.  2 root root        6 Aug 24 19:40 sys
drwxr-xr-x.  2 root root        6 Aug 24 19:40 sysroot
drwxr-xr-x.  2 root root        6 Aug 24 19:40 tmp
drwxr-xr-x.  7 root root       61 Aug 24 19:40 usr
drwxr-xr-x.  3 root root       47 Aug 24 19:40 var
# 看吧！上面几乎就像是一个小型的文件系统根目录耶！这样就能让 kernel 去挂载了！

# 4\. 接下来瞧一瞧到底这个小型的文件系统中，systemd 是要以哪个 target 来执行开机呢？
[root@study initramfs]# ll usr/lib/systemd/system/default.target
lrwxrwxrwx. 1 root root 13 Aug 24 19:40 usr/lib/systemd/system/default.target -&gt; initr

# 5\. 最终，让我们瞧一瞧系统内默认的 initrd.target 相依的所有服务数据吧！
[root@study initramfs]# systemctl list-dependencies initrd.target
initrd.target
├─dracut-cmdline.service
.....（中间省略）.....
├─basic.target
│ ├─alsa-restore.service
.....（中间省略）.....
│ ├─slices.target
│ │ ├─-.slice
│ │ └─system.slice
│ ├─sockets.target
│ │ ├─dbus.socket
.....（中间省略）.....
│ │ └─systemd-udevd-kernel.socket
│ ├─sysinit.target
│ │ ├─dev-hugepages.mount
.....（中间省略）.....
│ │ ├─local-fs.target
│ │ │ ├─-.mount
│ │ │ ├─boot.mount
.....（中间省略）.....
│ │ └─swap.target
│ │   ├─dev-centos-swap.swap
```

```
.....（中间省略）.....
│ │    └─dev-mapper-centos\x2dswap.swap
│ └─timers.target
│   └─systemd-tmpfiles-clean.timer
├─initrd-fs.target
└─initrd-root-fs.target
# 依旧通过 systemd 的方式，一个一个的将所有的侦测与服务载入系统中！
```

通过上面解开 initramfs 的结果，你会知道其实 initramfs 就是一个小型的根目录，这个小型根目录里面也是通过 systemd 来进行管理，同时观察 default.target 的链接，会发现其实这个小型系统就是通过 initrd.target 来开机，而 initrd.target 也是需要读入一堆例如 basic.target, sysinit.target 等等的硬件侦测、核心功能启用的流程， 然后开始让系统顺利运行。最终才又卸载 initramfs 的小型文件系统，实际挂载系统的根目录！

此外，initramfs 并没有包山包海，它仅是带入开机过程会用到的核心模块而已。所以如果你在 initramfs 里面去找 modules 这个关键字的话， 就可以发现主要的核心模块大概就是 SCSI、virtio、RAID 等等跟磁盘相关性比较高的模块就是了！现在由于磁盘大部分都是使用 SATA 这玩意儿， 并没有 IDE 的格式啰！所以，没有 initramfs 的话，你的 Linux 几乎就是不能顺利开机的啦！除非你将 SATA 的模块直接编译到核心去了！ ^_^

在核心完整的载入后，您的主机应该就开始正确的运行了，接下来，就是要开始执行系统的第一支程序： systemd ！

19.1.3 第一支程序 systemd 及使用 default.target 进入开机程序分析

在核心载入完毕、进行完硬件侦测与驱动程序载入后，此时你的主机硬件应该已经准备就绪了 （ready） ， 此时核心会主动的调用第一支程序，那就是 systemd 啰。这也是为啥第十六章的 pstree 指令介绍时，你会发现 systemd 的 PID 号码是一号啦。 systemd 最主要的功能就是准备软件执行的环境，包括系统的主机名称、网络设置、语系处理、文件系统格式及其他服务的启动等。 而所有的动作都会通过 systemd 的默认启动服务集合，亦即是 /etc/systemd/system/default.target 来规划。 另外， systemd 已经舍弃沿用多年的 system V 的 runlevel 了喔！

- 常见的操作环境 target 与相容于 runlevel 的等级

可以作为默认的操作环境 （default.target） 的主要项目有： multi-user.target 以及 graphical.target 这两个。当然还有某些比较特殊的操作环境， 包括在第十七章里面谈到的 rescue.target, emergency.target, shutdown.target 等等，以及本章在 initramfs 里面谈到的 initrd.target 啰！

但是过去的 systemV 使用的是一个称为 runlevel （执行等级） 的概念来启动系统的， systemd 为了相容于旧式的 systemV 操作行为， 所以也将 runlevel 与操作环境做个结合喔！你可以使用下面的方式来查询两者间的对应：

```
[root@study ~]# ll -d /usr/lib/systemd/system/runlevel*.target &#124; cut -c 28-
May  4 17:52 /usr/lib/systemd/system/runlevel0.target -&gt; poweroff.target
May  4 17:52 /usr/lib/systemd/system/runlevel1.target -&gt; rescue.target
May  4 17:52 /usr/lib/systemd/system/runlevel2.target -&gt; multi-user.target
May  4 17:52 /usr/lib/systemd/system/runlevel3.target -&gt; multi-user.target
May  4 17:52 /usr/lib/systemd/system/runlevel4.target -&gt; multi-user.target
May  4 17:52 /usr/lib/systemd/system/runlevel5.target -&gt; graphical.target
May  4 17:52 /usr/lib/systemd/system/runlevel6.target -&gt; reboot.target
```

如果你之前已经使用过 systemV 的方式来管理系统的话，那应该会知道切换执行等级可以使用“ init 3 ”转成文字界面，“ init 5 ”转成图形界面吧？ 这个 init 程序依旧是保留下来的，只是 init 3 会相当于 systemctl isolate multi-user.target 就是了！如果做个完整的叠代，这两个东西的对应为：

SystemV	systemd
init 0	systemctl poweroff
init 1	systemctl rescue
init [234]	systemctl isolate multi-user.target
init 5	systemctl isolate graphical.target
init 6	systemctl reboot

- systemd 的处理流程

如前所述，当我们取得了 /etc/systemd/system/default.target 这一个默认操作界面的设置之后，接下来系统帮我们做了什么呢？ 首先，它会链接到 /usr/lib/systemd/system/ 这个目录下去取得 multi-user.target 或 graphical.target 这两个其中的一 （当然， 鸟哥说的是正常的进入 Linux 操作环境的情况下！），假设我们是使用 graphical.target 好了，接着下来 systemd 会去找两个地方的设置， 就是如下的目录：

- /etc/systemd/system/graphical.target.wants/：使用者设置载入的 unit
- /usr/lib/systemd/system/graphical.target.wants/：系统默认载入的 unit

然后再由 /usr/lib/systemd/system/graphical.target 这个配置文件内发现如下的数据：

```
[root@study ~]# cat /usr/lib/systemd/system/graphical.target
[Unit]
Description=Graphical Interface
Documentation=man:systemd.special（7）
Requires=multi-user.target
After=multi-user.target
Conflicts=rescue.target
Wants=display-manager.service
AllowIsolate=yes

[Install]
Alias=default.target
```

这表示 graphical.target 必须要完成 multi-user.target 之后才能够进行，而进行完 graphical.target 之后，还得要启动 display-manager.service 才行的意思。 好了！那么通过同样的方式，我们来找找 multi-user.target 要执行完毕得要载入的项目有哪些呢？

```
# 先来看看 multi-user.target 配置文件内规范了相依的操作环境有哪些呢？
[root@study ~]# cat /usr/lib/systemd/system/multi-user.target
[Unit]
Description=Multi-User System
Documentation=man:systemd.special（7）
Requires=basic.target
Conflicts=rescue.service rescue.target
After=basic.target rescue.service rescue.target
AllowIsolate=yes

[Install]
Alias=default.target

# 然后看看系统默认要载入的 unit 有哪些？
[root@study ~]# ls /usr/lib/systemd/system/multi-user.target.wants
brandbot.path  plymouth-quit.service           systemd-logind.service
dbus.service   plymouth-quit-wait.service      systemd-user-sessions.service
getty.target   systemd-ask-password-wall.path

# 使用者自订要载入的 unit 又有哪些呢？
[root@study ~]# ls /etc/systemd/system/multi-user.target.wants
abrt-ccpp.service    crond.service          mdmonitor.service      sshd.service
abrtd.service        hypervkvpd.service     ModemManager.service   sysstat.service
abrt-oops.service    hypervvssd.service     NetworkManager.service tuned.service
abrt-vmcore.service  irqbalance.service     postfix.service        vmtoolsd.service
abrt-xorg.service    kdump.service          remote-fs.target       vsftpd2.service
atd.service          ksm.service            rngd.service           vsftpd.service
auditd.service       ksmtuned.service       rsyslog.service
backup2.timer        libstoragemgmt.service smartd.service
backup.timer         libvirtd.service       sshd2.service
```

通过上面的结果，我们又能知道 multi-usre.target 需要在 basic.target 运行完毕才能够载入上述的许多 unit 哩！然后再去 basic.target 里头找数据等等～ 最终这些数据就可以通过“ systemctl list-dependencies graphical.target ”这个指令来列出所有的相关性的服务啰！这就是 systemd 的调用所需要的服务的流程喔！



Tips 要知道系统的服务启用的流程，最简单的方法就是“ systemctl list-dependencies graphical.target ”这个指令！只是，如果你想要知道背后的配置文件意义， 那就是分别去找出 /etc 与 /usr/lib 下面的 graphical.target.wants/ 目录下的数据就对了！当然，配置文件脚本里面的 Requires 这个设置值所代表的服务， 也是需要是先载入喔！

约略分析一下“ systemctl list-dependencies graphical.target ”所输出的相依属性服务，基本上我们 CentOS 7.x 的 systemd 开机流程大约是这样：

1. local-fs.target + swap.target：这两个 target 主要在挂载本机 /etc/fstab 里面所规范的文件系统与相关的内存交换空间。

2. sysinit.target：这个 target 主要在侦测硬件，载入所需要的核心模块等动作。
3. basic.target：载入主要的周边硬件驱动程序与防火墙相关任务
4. multi-user.target 下面的其它一般系统或网络服务的载入
5. 图形界面相关服务如 gdm.service 等其他服务的载入

除了第一步骤 local-fs.target, swap.target 是通过 /etc/fstab 来进行挂载的行为之外，那其他的 target 有做啥动作呢？简单得来说说！

19.1.4 systemd 执行 sysinit.target 初始化系统、basic.target 准备系统

如果你自己使用“ systemctl list-dependencies sysinit.target ”来瞧瞧的话，那就会看到很多相依的服务！这些服务你应该要一个一个去查询看看设置脚本的内容， 就能够大致理解每个服务的意义。基本上，我们可以将这些服务归类成几个大项就是了：

- 特殊文件系统设备的挂载：包括 dev-hugepages.mount dev-mqueue.mount 等挂载服务，主要在挂载跟巨量内存分页使用与讯息伫列的功能。 挂载成功后，会在 /dev 下面创建 /dev/hugepages/, /dev/mqueue/ 等目录；
- 特殊文件系统的启用：包括磁盘阵列、网络磁盘 （iscsi）、LVM 文件系统、文件系统对照服务 （multipath） 等等，也会在这里被侦测与使用到！
- 开机过程的讯息传递与动画执行：使用 plymouthd 服务搭配 plymouth 指令来传递动画与讯息
- 日志式登录文件的使用：就是 systemd-journald 这个服务的启用啊！
- 载入额外的核心模块：通过 /etc/modules-load.d/*.conf 文件的设置，让核心额外载入管理员所需要的核心模块！
- 载入额外的核心参数设置：包括 /etc/sysctl.conf 以及 /etc/sysctl.d/*.conf 内部设置！
- 启动系统的乱数产生器：乱数产生器可以帮助系统进行一些密码加密演算的功能
- 设置终端机 （console） 字形
- 启动动态设备管理员：就是 udevd 这个家伙！用在动态对应实际设备存取与设备文件名对应的一个服务！相当重要喔！也是在这里启动的！

不论你即将使用哪种操作环境来使用系统，这个 sysinit.target 几乎都是必要的工作！从上面你也可以看的出来，基本的核心功能、文件系统、文件系统设备的驱动等等， 都在这个时刻处理完毕～所以，这个 sysinit.target 的阶段是挺重要的喔！

执行完 sysinit.target 之后，再来则是 basic.target 这个项目了。 sysinit.target 在初始化系统，而这个 basic .target 则是一个最阳春的操作系统了！ 这个 basic.target 的阶段主要启动的服务大概有这些：

- 载入 alsa 音效驱动程序：这个 alsa 是个音效相关的驱动程序，会让你的系统有音效产生啰；
- 载入 firewalld 防火墙：CentOS 7.x 以后使用 firewalld 取代 iptables 的防火墙设置，虽然最终都是使用 iptables 的架构， 不过在设置上面差很多喔！

- 载入 CPU 的微指令功能；
- 启动与设置 SELinux 的安全本文：如果由 disable 的状态改成 enable 的状态，或者是管理员设置强制重新设置一次 SELinux 的安全本文，也在这个阶段处理喔！
- 将目前的开机过程所产生的开机信息写入到 /var/log/dmesg 当中
- 由 /etc/sysconfig/modules/*.modules 及 /etc/rc.modules 载入管理员指定的模块！
- 载入 systemd 支持的 timer 功能；

在这个阶段完成之后，你的系统已经可以顺利的运行！就差一堆你需要的登陆服务、网络服务、本机认证服务等等的 service 类别啰！于是就可以进入下个服务启动的阶段了！

19.1.5 systemd 启动 multi-user.target 下的服务

在载入核心驱动硬件后，经过 sysinit.target 的初始化流程让系统可以存取之后，加上 basic.target 让系统成为操作系统的基础，之后就是服务器要顺利运行时，需要的各种主机服务以及提供服务器功能的网络服务的启动了。这些服务的启动则大多是附挂在 multi-user.target 这个操作环境下面，你可以到 /etc/systemd/system/multi-user.target.wants/ 里头去瞧瞧默认要被启动的服务喔！

也就是说，一般来说服务的启动脚本设置都是放在下面的目录内：

- /usr/lib/systemd/system （系统默认的服务启动脚本设置）
- /etc/systemd/system （管理员自己开发与设置的脚本设置）

而使用者针对主机的本机服务与服务器网络服务的各项 unit 若要 enable 的话，就是将它放到 /etc/systemd/system/multi-user.target.wants/ 这个目录下面做个链接～ 这样就可以在开机的时候去启动他。这时回想一下，你在第十七章使用 systemctl enable/disable 时，系统的回应是什么呢？再次回想一下：

```
# 将 vsftpd.service 先 disable 再 enable 看看输出的信息为何？
[root@study ~]# systemctl disable vsftpd.service
rm '/etc/systemd/system/multi-user.target.wants/vsftpd.service'

[root@study ~]# systemctl enable vsftpd.service
ln -s '/usr/lib/systemd/system/vsftpd.service' '/etc/systemd/system/multi-user.target.
 wants/vsftpd.service'
```

有没有发现亮点了？不是从 /etc/systemd/system/multi-user.target.wants/ 里面删除链接文件，就是创建链接文件～这样说，理解吧？ 你当然不需要手动作这些链接，而是使用 systemctl 来处理即可！另外，这些程序除非在脚本设置里面原本就有规范服务的相依性， 这样才会有顺序的启动之外，大多数的服务都是同时启动的！这就是 systemd 的多任务啰。

- 相容 systemV 的 rc-local.service

另外，过去用过 Linux 的朋友大概都知道，当系统完成开机后，还想要让系统额外执行某些程序的话，可以将该程序指令或脚本的绝对路径名称写入到 /etc/rc.d/rc.local 这个文件去！新的 systemd 机制中，它建议直接写一个 systemd 的启动脚本配置文件到 /etc/systemd/system

下面，然后使用 systemctl enable 的方式来设置启用它，而不要直接使用 rc.local 这个文件啦！

但是像鸟哥这种老人家就是喜欢将开机后要立刻执行的许多管理员自己的脚本，将它写入到 /etc/rc.d/rc.local 去嘛！那新版的 systemd 有没有支持呢？ 当然有！那就是 rc-local.service 这个服务的功能了！这个服务不需要启动，它会自己判断 /etc/rc.d/rc.local 是否具有可执行的权限来判断要不要启动这个服务！ 你可以这样检查看看：

```
# 1\. 先看一下 /etc/rc.d/rc.local 的权限，然后检查 multi-user.target 有没有这个服务
[root@study ~]# ll /etc/rc.d/rc.local
-rw-r--r--. 1 root root 473 Mar  6 13:48 /etc/rc.d/rc.local

[root@study ~]# systemctl status rc-local.service
rc-local.service - /etc/rc.d/rc.local Compatibility
   Loaded: loaded （/usr/lib/systemd/system/rc-local.service; static）
   Active: inactive （dead）

[root@study ~]# systemctl list-dependencies multi-user.target &#124; grep rc-local
# 明明就有这个服务，但是 rc.local 不具有可执行 （x） 的权限，因此这个服务不会被执行

# 2\. 加入可执行权限后，再看一下 rc-local 是否可被启用！
[root@study ~]# chmod a+x /etc/rc.d/rc.local; ll /etc/rc.d/rc.local
-rwxr-xr-x. 1 root root 473 Mar  6 13:48 /etc/rc.d/rc.local

[root@study ~]# systemctl daemon-reload
[root@study ~]# systemctl list-dependencies multi-user.target &#124; grep rc-local
├─rc-local.service   # 这个服务确实被记录到启动的环境下啰！
```

通过这个 chmod a+x /etc/rc.d/rc.local 的步骤，你的许多脚本就可以放在 /etc/rc.d/rc.local 这个文件内， 系统在每次开机都会去执行这文件内的指令喔！非常简单吧！

- 提供 tty 界面与登陆的服务

在 multi-user.target 下面还有个 getty.target 的操作界面项目喔！ 这个项目就是我们在第十七章用来举例的 tty 终端机界面的个数案例。 能不能提供适当的登陆服务也是 multi-user.target 下面的内容！包括 systemd-logind.service, systemd-user-sessions.service 等服务。

比较有趣的地方是，由于服务都是同步运行，不一定哪个服务先启动完毕。如果 getty 服务先启动完毕时，你会发现到有可用的终端机尝试让你登陆系统了。 问题是，如果 systemd-logind.service 或 systemd-user-sessions.service 服务尚未执行完毕的话，那么你还是无法登陆系统的。



Tips 有些比较急性子的伙伴在启动 CentOS 7.x 时，看到屏幕出现 tty1 可以让他登陆了～但是一开始输入正确的帐密却无法登陆系统！ 总要隔了数十秒之后才能够顺利的登陆！知道原因了吗？ ^_^

19.1.6 systemd 启动 graphical.target 下面的服务

如果你的 default.target 是 multi-user.target 的话，那么这个步骤就不会进行。反之，如果是 graphical.target 的话，那么 systemd 就会开始载入用户管理服务与图形界面管理员（window display manager, DM） 等，启动图形界面来让用户以图形界面登陆系统喔！ 如果你对于 graphical.target 多了哪些服务有兴趣，那就来检查看看：

```
[root@study ~]# systemctl list-dependencies graphical.target
graphical.target
├─accounts-daemon.service
├─gdm.service
├─network.service
├─rtkit-daemon.service
├─systemd-update-utmp-runlevel.service
└─multi-user.target
  ├─abrt-ccpp.service
.....（下面省略）.....
```

事实上就是多了上面列出来的这些服务而已～大多数都是图形界面帐号管理的功能，至于实际让用户可以登陆的服务，倒是那个 gdm.service 哩！ 如果你去瞧瞧 gdm.service 的内容，就会发现最重要的可执行文件是 /usr/sbin/gdm 喔！那就是让使用者可以利用图形界面登陆的最重要服务啰！ 我们未来讲到 X 窗口界面时再来聊聊 gdm 这玩意儿喔！

到此为止，systemd 就已经完整的处理完毕，你可以使用图形界面或文字界面的方式来登陆系统，系统也顺利的开机完毕， 也能够将你写入到 /etc/rc.d/rc.local 的脚本实际执行一次啰。那如果默认是图形界面 （graphical.target） 但是想要关掉而进入文字界面 （multi-user.target） 呢？ 很简单啊！19.1.3小节就谈过了，使用“ systemctl isolate multi-user.target ”即可！如果使用“ init 3 ”呢？也是可以啦！ 只是系统实际执行的还是“ systemctl isolate multi-user.target ”就是了！ ^_^

19.1.7 开机过程会用到的主要配置文件

基本上， systemd 有自己的配置文件处理方式，不过为了相容于 systemV ，其实很多的服务脚本设置还是会读取位于 /etc/sysconfig/ 下面的环境配置文件！ 下面我们就来谈谈几个常见的比较重要的配置文件啰！

- 关于模块： /etc/modprobe.d/*.conf* 及 */etc/modules-load.d/*.conf

还记得我们在 sysinit.target 系统初始化 当中谈到的载入使用者自订模块的地方吗？其实有两个地方可以处理模块载入的问题，包括：

- /etc/modules-load.d/*.conf：单纯要核心载入模块的位置；
- /etc/modprobe.d/*.conf：可以加上模块参数的位置

基本上 systemd 已经帮我们将开机会用到的驱动程序全部载入了，因此这个部份你应该无须更动才对！不过， 如果你有某些特定的参数要处理时，应该就得要在这里进行了。举例来说，我们在第十七章曾经谈过 vsftpd 这个服务对吧！ 而且当时将这个服务的端口更改到 555

这个号码上去了！那我们可能需要修改防火墙设置，其中一个针对 FTP 很重要的防火墙模块为 nf_conntrack_ftp， 因此，你可以将这个模块写入到系统开机流程中，例如：

```
[root@study ~]# vim /etc/modules-load.d/vbird.conf
nf_conntrack_ftp
```

一个模块 （驱动程序） 写一行～然后，上述的模块基本上是针对默认 FTP 端口，亦即 port 21 所设置的，如果需要调整到 port 555 的话， 得要外带参数才行！模块外加参数的设置方式得要写入到另一个地方喔！

```
[root@study ~]# vim /etc/modprobe.d/vbird.conf
options nf_conntrack_ftp ports=555
```

之后重新开机就能够顺利的载入并且处理好这个模块了。不过，如果你不想要开机测试，想现在处理呢？有个方式可以来进行看看：

```
[root@study ~]# lsmod &#124; grep nf_conntrack_ftp
# 没东西！因为还没有载入这个模块！所以不会出现任何讯息！

[root@study ~]# systemctl restart systemd-modules-load.service
[root@study ~]# lsmod &#124; grep nf_conntrack_ftp
nf_conntrack_ftp       18638  0
nf_conntrack          105702  1 nf_conntrack_ftp
```

通过上述的方式，你就可以在开机的时候将你所需要的驱动程序载入或者是调整这些模块的外加参数啰！

- /etc/sysconfig/*

还有哪些常见的环境配置文件呢？我们找几个比较重要的来谈谈：

- authconfig： 这个文件主要在规范使用者的身份认证的机制，包括是否使用本机的 /etc/passwd, /etc/shadow 等， 以及 /etc/shadow 密码记录使用何种加密演算法，还有是否使用外部密码服务器提供的帐号验证 （NIS, LDAP） 等。 系统默认使用 SHA512 加密演算法，并且不使用外部的身份验证机制；另外，不建议手动修改这个文件喔！你应该使用“ authconfig-tui ”指令来修改较佳！
- cpupower： 如果你有启动 cpupower.service 服务时，他就会读取这个配置文件。主要是 Linux 核心如何操作 CPU 的原则。 一般来说，启动 cpupower.service 之后，系统会让 CPU 以最大性能的方式来运行，否则默认就是用多少算多少的模式来处理的。
- firewalld, iptables-config, iptables-config, ebtables-config： 与防火墙服务的启动外带的参数有关，这些数据我们会在服务器篇慢慢再来讨论。
- network-scripts/： 至于 network-scripts 里面的文件，则是主要用在设置网卡～ 这部份我们在服务器架设篇才会提到！

19.2 核心与核心模块

谈完了整个开机的流程，您应该会知道，在整个开机的过程当中，是否能够成功的驱动我们主机的硬件配备， 是核心 （kernel） 的工作！而核心一般都是压缩文件，因此在使用核心之前，就得要将他解压缩后，才能载入内存当中。

另外，为了应付日新月异的硬件，目前的核心都是具有“可读取模块化驱动程序”的功能， 亦即是所谓的“ modules （模块化）”的功能啦！所谓的模块化可以将他想成是一个“外挂程序”，该外挂程序可能由硬件开发厂商提供，也有可能我们的核心本来就支持～不过，较新的硬件， 通常都需要硬件开发商提供驱动程序模块啦！

那么核心与核心模块放在哪？

- 核心： /boot/vmlinuz 或 /boot/vmlinuz-version；
- 核心解压缩所需 RAM Disk： /boot/initramfs （/boot/initramfs-version）；
- 核心模块： /lib/modules/version/kernel 或 /lib/modules/$（uname -r）/kernel；
- 核心源代码： /usr/src/linux 或 /usr/src/kernels/ （要安装才会有，默认不安装）

如果该核心被顺利的载入系统当中了，那么就会有几个信息纪录下来：

- 核心版本： /proc/version
- 系统核心功能： /proc/sys/kernel/

问题来啦，如果我有个新的硬件，偏偏我的操作系统不支持，该怎么办？很简单啊！

- 重新编译核心，并加入最新的硬件驱动程序源代码；
- 将该硬件的驱动程序编译成为模块，在开机时载入该模块

上面第一点还很好理解，反正就是重新编译核心就是了。不过，核心编译很不容易啊！ 我们会在后续章节约略介绍核心编译的整个程序。比较有趣的则是将该硬件的驱动程序编译成为模块啦！ 关于编译的方法，可以参考后续的第二十一章、源代码与 tarball的介绍。 我们这个章节仅是说明一下，如果想要载入一个已经存在的模块时，该如何是好？

19.2.1 核心模块与相依性

既然要处理核心模块，自然就得要了解了解我们核心提供的模块之间的相关性啦！ 基本上，核心模块的放置处是在 /lib/modules/$（uname -r）/kernel 当中，里面主要还分成几个目录：

```
arch      ：与硬件平台有关的项目，例如 CPU 的等级等等；
crypto    ：核心所支持的加密的技术，例如 md5 或者是 des 等等；
drivers   ：一些硬件的驱动程序，例如显卡、网卡、PCI 相关硬件等等；
fs        ：核心所支持的 filesystems ，例如 vfat, reiserfs, nfs 等等；
lib       ：一些函数库；
net       ：与网络有关的各项协定数据，还有防火墙模块 （net/ipv4/netfilter/*） 等等；
sound     ：与音效有关的各项模块；
```

如果要我们一个一个的去检查这些模块的主要信息，然后定义出他们的相依性， 我们可能会疯掉吧！所以说，我们的 Linux 当然会提供一些模块相依性的解决方案啰～ 对啦！那就是检查 /lib/modules/$（uname -r）/modules.dep 这个文件啦！他记录了在核心支持的模块的各项相依性。

那么这个文件如何创建呢？挺简单！利用 depmod 这个指令就可以达到创建该文件的需求了！

```
[root@study ~]# depmod [-Ane]
选项与参数：
-A  ：不加任何参数时， depmod 会主动的去分析目前核心的模块，并且重新写入
      /lib/modules/$（uname -r）/modules.dep 当中。若加入 -A 参数时，则 depmod
      会去搜寻比 modules.dep 内还要新的模块，如果真找到新模块，才会更新。
-n  ：不写入 modules.dep ，而是将结果输出到屏幕上（standard out）；
-e  ：显示出目前已载入的不可执行的模块名称

范例一：若我做好一个网卡驱动程序，文件名为 a.ko，该如何更新核心相依性？
[root@study ~]# cp a.ko /lib/modules/$（uname -r）/kernel/drivers/net
[root@study ~]# depmod
```

以上面的范例一为例，我们的 kernel 核心模块扩展名一定是 .ko 结尾的， 当你使用 depmod 之后，该程序会跑到模块标准放置目录 /lib/modules/$（uname -r）/kernel ， 并依据相关目录的定义将全部的模块捉出来分析，最终才将分析的结果写入 modules.dep 文件中的呐！ 这个文件很重要喔！因为他会影响到本章稍后会介绍的 modprobe 指令的应用！

19.2.2 核心模块的观察

那你到底晓不晓得目前核心载入了多少的模块呢？粉简单啦！利用 lsmod 即可！

```
[root@study ~]# lsmod
Module                  Size  Used by
nf_conntrack_ftp       18638  0
nf_conntrack          105702  1 nf_conntrack_ftp
....（中间省略）....
qxl                    73766  1
drm_kms_helper         98226  1 qxl
ttm                    93488  1 qxl
drm                   311588  4 qxl,ttm,drm_kms_helper  # drm 还被 qxl, ttm..等模块使用
....（下面省略）....
```

使用 lsmod 之后，系统会显示出目前已经存在于核心当中的模块，显示的内容包括有：

- 模块名称（Module）；
- 模块的大小（size）；
- 此模块是否被其他模块所使用 （Used by）。

也就是说，模块其实真的有相依性喔！举上表为例，nf_conntrack 先被载入后，nf_conntrack_ftp这个模块才能够进一步的载入系统中！ 这两者间是有相依性的。包括鸟哥测试机使用的是虚拟机，用到的显卡是 qxl 这个模块，该模块也同时使用了好多额外的附属模块喔！ 那么，那个 drm 是啥鬼？要如何了解呢？就用 modinfo 吧！

```
[root@study ~]# modinfo [-adln] [module_name&#124;filename]
选项与参数：
-a  ：仅列出作者名称；
-d  ：仅列出该 modules 的说明 （description）；
-l  ：仅列出授权 （license）；
-n  ：仅列出该模块的详细路径。

范例一：由上个表格当中，请列出 drm 这个模块的相关信息：
[root@study ~]# modinfo drm
filename:       /lib/modules/3.10.0-229.el7.x86_64/kernel/drivers/gpu/drm/drm.ko
license:        GPL and additional rights
description:    DRM shared core routines
author:         Gareth Hughes, Leif Delgass, José Fonseca, Jon Smirl
rhelversion:    7.1
srcversion:     66683E37FDD905C9FFD7931
depends:        i2c-core
intree:         Y
vermagic:       3.10.0-229.el7.x86_64 SMP mod_unload modversions
signer:         CentOS Linux kernel signing key
sig_key:        A6:2A:0E:1D:6A:6E:48:4E:9B:FD:73:68:AF:34:08:10:48:E5:35:E5
sig_hashalgo:   sha256
parm:           edid_fixup:Minimum number of valid EDID header Bytes （0-8, default 6） （
.....（下面省略）.....
# 可以看到这个模块的来源，以及该模块的简易说明！

范例二：我有一个模块名称为 a.ko ，请问该模块的信息为？
[root@study ~]# modinfo a.ko
....（省略）....
```

事实上，这个 modinfo 除了可以“查阅在核心内的模块”之外，还可以检查“某个模块文件”， 因此，如果你想要知道某个文件代表的意义为何，利用 modinfo 加上完整文件名吧！看看就晓得是啥玩意儿啰！ ^_^

19.2.3 核心模块的载入与移除

好了，如果我想要自行手动载入模块，又该如何是好？有很多方法啦，最简单而且建议的，是使用 modprobe 这个指令来载入模块， 这是因为 modprobe 会主动的去搜寻 modules.dep 的内容，先克服了模块的相依性后， 才决定需要载入的模块有哪些，很方便。至于 insmod 则完全由使用者自行载入一个完整文件名的模块， 并不会主动的分析模块相依性啊！

```
[root@study ~]# insmod [/full/path/module_name] [parameters]

范例一：请尝试载入 cifs.ko 这个“文件系统”模块
[root@study ~]# insmod /lib/modules/$（uname -r）/kernel/fs/fat/fat.ko
[root@study ~]# lsmod &#124; grep fat
fat                    65913  0
```

insmod 立刻就将该模块载入啰～但是 insmod 后面接的模块必须要是完整的“文件名”才行！ 那如何移除这个模块呢？

```
[root@study ~]# rmmod [-fw] module_name
选项与参数：
-f  ：强制将该模块移除掉，不论是否正被使用；

范例一：将刚刚载入的 fat 模块移除！
[root@study ~]# rmmod fat

范例二：请载入 vfat 这个“文件系统”模块
[root@study ~]# insmod /lib/modules/$（uname -r）/kernel/fs/vfat/vfat.ko
insmod: ERROR: could not load module /lib/modules/3.10.0-229.el7.x86_64/kernel/fs/vfat/
 vfat.ko: No such file or directory
# 无法载入 vfat 这个模块啊！伤脑筋！
```

使用 insmod 与 rmmod 的问题就是，你必须要自行找到模块的完整文件名才行，而且如同上述范例二的结果， 万一模块有相依属性的问题时，你将无法直接载入或移除该模块呢！所以近年来我们都建议直接使用 modprobe 来处理模块载入的问题，这个指令的用法是：

```
[root@study ~]# modprobe [-cfr] module_name
选项与参数：
-c  ：列出目前系统所有的模块！（更详细的代号对应表）
-f  ：强制载入该模块；
-r  ：类似 rmmod ，就是移除某个模块啰～

范例一：载入 vfat 模块
[root@study ~]# modprobe vfat
# 很方便吧！不需要知道完整的模块文件名，这是因为该完整文件名已经记录到
# /lib/modules/`uname -r`/modules.dep 当中的缘故啊！如果要移除的话：
[root@study ~]# modprobe -r vfat
```

使用 modprobe 真的是要比 insmod 方便很多！因为他是直接去搜寻 modules.dep 的纪录，所以啰，当然可以克服模块的相依性问题，而且还不需要知道该模块的详细路径呢！好方便！ ^_^

例题：尝试使用 modprobe 载入 cifs 这个模块，并且观察该模块的相关模块是哪个？答：我们使用 modprobe 来载入，再以 lsmod 来观察与 grep 撷取关键字看看：

```
[root@study ~]# modprobe cifs
[root@study ~]# lsmod &#124; grep cifs
cifs                  456500  0
dns_resolver           13140  1 cifs   &lt;==竟然还有使用到 dns_resolver 哩！

[root@study ~]# modprobe -r cifs &lt;==测试完移除此模块
```

19.2.4 核心模块的额外参数设置：/etc/modprobe.d/*conf

如果有某些特殊的需求导致你必须要让核心模块加上某些参数时，请回到19.1.7小节瞧一瞧！应该会有启发喔！重点就是要自己创建扩展名为 .conf 的文件，通过 options 来带入核心模块参数啰！

19.3 Boot Loader: Grub2

在看完了前面的整个开机流程，以及核心模块的整理之后，你应该会发现到一件事情，那就是“ boot loader 是载入核心的重要工具”啊！没有 boot loader 的话，那么 kernel 根本就没有办法被系统载入的呢！所以，下面我们会先谈一谈 boot loader 的功能，然后再讲一讲现阶段 Linux 里头最主流的 grub2 这个 boot loader 吧！

另外，你也得要知道，目前新版的 CentOS 7.x 已经将沿用多年的 grub 换成了 grub2 了！这个 grub2 版本在设置与安装上面跟之前的 grub 有点不那么相同， 所以，在后续的章节中，得要了解一下新的 grub2 的设置方式才行喔！如果你是新接触者，那没关系～直接看就 OK 了！

19.3.1 boot loader 的两个 stage

我们在第一小节开机流程的地方曾经讲过，在 BIOS 读完信息后，接下来就是会到第一个开机设备的 MBR 去读取 boot loader 了。这个 boot loader 可以具有菜单功能、直接载入核心文件以及控制权移交的功能等， 系统必须要有 loader 才有办法载入该操作系统的核心就是了。但是我们都知道， MBR 是整个硬盘的第一个 sector 内的一个区块，充其量整个大小也才 446 Bytes 而已。即使是 GPT 也没有很大的扇区来储存 loader 的数据。 我们的 loader 功能这么强，光是程序码与设置数据不可能只占这么一点点的容量吧？那如何安装？

为了解决这个问题，所以 Linux 将 boot loader 的程序码执行与设置值载入分成两个阶段 （stage） 来执行：

- Stage 1：执行 boot loader 主程序： 第一阶段为执行 boot loader 的主程序，这个主程序必须要被安装在开机区，亦即是 MBR 或者是 boot sector 。但如前所述，因为 MBR 实在太小了，所以，MBR 或 boot sector 通常仅安装 boot loader 的最小主程序， 并没有安装 loader 的相关配置文件；
- Stage 2：主程序载入配置文件： 第二阶段为通过 boot loader 载入所有配置文件与相关的环境参数文件 （包括文件系统定义与主要配置文件 grub.cfg）， 一般来说，配置文件都在 /boot 下面。

那么这些配置文件是放在哪里啊？这些与 grub2 有关的文件都放置到 /boot/grub2 中，那我们就来看看有哪些文件吧！

```
[root@study ~]# ls -l /boot/grub2
-rw-r--r--.  device.map            &lt;==grub2 的设备对应档（下面会谈到）
drwxr-xr-x.  fonts                 &lt;==开机过程中的画面会使用到的字体数据
-rw-r--r--.  grub.cfg              &lt;==grub2 的主配置文件！相当重要！
-rw-r--r--.  grubenv               &lt;==一些环境区块的符号
drwxr-xr-x.  i386-pc               &lt;==针对一般 x86 PC 所需要的 grub2 的相关模块
drwxr-xr-x.  locale                &lt;==就是语系相关的数据啰
drwxr-xr-x.  themes                &lt;==一些开机主题画面数据

[root@study ~]# ls -l /boot/grub2/i386-pc
-rw-r--r--.  acpi.mod              &lt;==电源管理有关的模块
-rw-r--r--.  ata.mod               &lt;==磁盘有关的模块
-rw-r--r--.  chain.mod             &lt;==进行 loader 控制权移交的相关模块
-rw-r--r--.  command.lst           &lt;==一些指令相关性的列表
-rw-r--r--.  efiemu32.o            &lt;==下面几个则是与 uefi BIOS 相关的模块
-rw-r--r--.  efiemu64.o
-rw-r--r--.  efiemu.mod
-rw-r--r--.  ext2.mod              &lt;==EXT 文件系统家族相关模块
-rw-r--r--.  fat.mod               &lt;==FAT 文件系统模块
-rw-r--r--.  gcry_sha256.mod       &lt;==常见的加密模块
-rw-r--r--.  gcry_sha512.mod
-rw-r--r--.  iso9660.mod           &lt;==光盘文件系统模块
-rw-r--r--.  lvm.mod               &lt;==LVM 文件系统模块
-rw-r--r--.  mdraid09.mod          &lt;==软件磁盘阵列模块
-rw-r--r--.  minix.mod             &lt;==MINIX 相关文件系统模块
-rw-r--r--.  msdospart.mod         &lt;==一般 MBR 分区表
-rw-r--r--.  part_gpt.mod          &lt;==GPT 分区表
-rw-r--r--.  part_msdos.mod        &lt;==MBR 分区表
-rw-r--r--.  scsi.mod              &lt;==SCSI 相关模块
-rw-r--r--.  usb_keyboard.mod      &lt;==下面两个为 USB 相关模块
-rw-r--r--.  usb.mod
-rw-r--r--.  vga.mod               &lt;==VGA 显卡相关模块
-rw-r--r--.  xfs.mod               &lt;==XFS 文件系统模块
# 鸟哥这里只拿一些模块作说明，没有全部的文件都列上来喔！
```

从上面的说明你可以知道 /boot/grub2/ 目录下最重要的就是配置文件 （grub2.cfg） 以及各种文件系统的定义！ 我们的 loader 读取了这种文件系统定义数据后，就能够认识文件系统并读取在该文件系统内的核心文件啰。

所以从上面的文件来看， grub2 认识的文件系统与磁盘分区格式真的非常多喔！正因为如此，所以 grub2 才会取代 Lilo / grub 这个老牌的 boot loader 嘛！好了，接下来就来瞧瞧配置文件内有啥设置值吧！

19.3.2 grub2 的配置文件 /boot/grub2/grub.cfg 初探

grub2 的优点挺多的，包括有：

- 认识与支持较多的文件系统，并且可以使用 grub2 的主程序直接在文件系统中搜寻核心文件名；
- 开机的时候，可以“自行编辑与修改开机设置项目”，类似 bash 的指令模式；
- 可以动态搜寻配置文件，而不需要在修改配置文件后重新安装 grub2 。亦即是我们只要修改完 /boot/grub2/grub.cfg 里头的设置后，下次开机就生效了！

上面第三点其实就是 Stage 1, Stage 2 分别安装在 MBR （主程序） 与文件系统当中 （配置文件与定义文件） 的原因啦！ 好了，接下来，让我们好好了解一下 grub2 的配置文件：/boot/grub2/grub.cfg 这玩意儿吧！

- 磁盘与分区在 grub2 中的代号

安装在 MBR 的 grub2 主程序，最重要的任务之一就是从磁盘当中载入核心文件， 以让核心能够顺利的驱动整个系统的硬件。所以啰， grub2 必须要认识硬盘才行啊！那么 grub2 到底是如何认识硬盘的呢？ 嘿嘿！ grub2 对硬盘的代号设置与传统的 Linux 磁盘代号可完全是不同的！grub2 对硬盘的识别使用的是如下的代号：

```
(hd0,1)         # 一般的默认语法，由 grub2 自动判断分区格式
(hd0,msdos1)    # 此磁盘的分区为传统的 MBR 模式
(hd0,gpt1)      # 此磁盘的分区为 GPT 模式
```

够神了吧？跟 /dev/sda1 风马牛不相干～怎么办啊？其实只要注意几个东西即可，那就是：

- 硬盘代号以小括号 （ ） 包起来；
- 硬盘以 hd 表示，后面会接一组数字；
- 以“搜寻顺序”做为硬盘的编号！（这个重要！）
- 第一个搜寻到的硬盘为 0 号，第二个为 1 号，以此类推；
- 每颗硬盘的第一个 partition 代号为 1 ，依序类推。

所以说，第一颗“搜寻到的硬盘”代号为：“（hd0）”，而该颗硬盘的第一号分区为“（hd0,1）”，这样说了解了吧？ 另外，为了区分不同的分区格式，因此磁盘后面的分区号码可以使用类似 msdos1 与 gpt1 的方式来调整！ 最终要记得的是，磁盘的号码是由 0 开始编号，分区的号码则与 Linux 一样，是由 1 号开始编号！两者不同喔！



Tips 跟旧版的 grub 有点不一样，因为旧版的 grub 不论磁盘还是分区的起始号码都是 0 号，而 grub2 在分区的部份是以 1 号开始编喔！ 此外，由于 BIOS 可以调整磁盘的开机顺序，因此上述的磁盘对应的 （hdN） 那个号码 N 是可能会变动的喔！这要先有概念才行！

所以说，整个硬盘代号为：

硬盘搜寻顺序	在 **Grub2** 当中的代号
第一颗（MBR）	（hd0） （hd0,msdos1） （hd0,msdos2） （hd0,msdos3）....
第二颗（GPT）	（hd1） （hd1,gpt1） （hd1,gpt2） （hd1,gpt3）....
第三颗	（hd2） （hd2,1） （hd2,2） （hd2,3）....

这样应该比较好看出来了吧？第一颗硬盘的 MBR 安装处的硬盘代号就是“（hd0）”， 而第一颗硬盘的第一个分区的 boot sector 代号就是“（hd0,msdos1）”第一颗硬盘的第一个逻辑分区的 boot sector 代号为“（hd0,msdos5）”瞭了吧！

例题：假设你的系统仅有一颗 SATA 硬盘，请说明该硬盘的第一个逻辑分区在 Linux 与 grub2 当中的文件名与代号：答：因为是 SATA 磁盘，加上使用逻辑分区，因此 Linux 当中的文件名为 /dev/sda5 才对 （1~4 保留给 primary 与 extended 使用）。 至于 grub2 当中的磁盘代号则由于仅有一颗磁盘，因此代号会是“（hd0,msdos5）”或简易的写法“（hd0,5）”才对。

- /boot/grub2/grub.cfg 配置文件（重点在了解，不要随便改！）：

了解了 grub2 当中最麻烦的硬盘代号后，接下来，我们就可以瞧一瞧配置文件的内容了。先看一下鸟哥的 CentOS 内的 /boot/grub2/grub.cfg 好了：

```
[root@study ~]# vim /boot/grub2/grub.cfg
# 开始是 /etc/grub.d/00_header 这个脚本执行的结果展示，主要与基础设置与环境有关
### BEGIN /etc/grub.d/00_header ###
set pager=1

if [ -s $prefix/grubenv ]; then
  load_env
fi
.....（中间省略）.....
if [ x$feature_timeout_style = xy ] ; then
  set timeout_style=menu
 set timeout=5
# Fallback normal timeout code in case the timeout_style feature is
# unavailable.
else
 set timeout=5
fi
### END /etc/grub.d/00_header ###

# 开始执行 /etc/grub.d/10_linux，主要针对实际的 Linux 核心文件的开机环境
### BEGIN /etc/grub.d/10_linux ###
menuentry 'CentOS Linux 7 （Core）, with Linux 3.10.0-229.el7.x86_64' --class rhel fedora
  --class gnu-linux --class gnu --class os --unrestricted $menuentry_id_option  \
  'gnulinux-3.10.0-229.el7.x86_64-advanced-299bdc5b-de6d-486a-a0d2-375402aaab27' {
        load_video
        set gfxpayload=keep
        insmod gzio
        insmod part_gpt
        insmod xfs
        set root='hd0,gpt2'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint='hd0,gpt2'  94ac5f77-cb8a-495e-a
        else
          search --no-floppy --fs-uuid --set=root 94ac5f77-cb8a-495e-a65b-2ef7442b837c
        fi
        linux16 /vmlinuz-3.10.0-229.el7.x86_64 root=/dev/mapper/centos-root ro  \
                rd.lvm.lv=centos/root rd.lvm.lv=centos/swap crashkernel=auto rhgb quiet \
                LANG=zh_TW.UTF-8
        initrd16 /initramfs-3.10.0-229.el7.x86_64.img
}
### END /etc/grub.d/10_linux ###
.....（中间省略）.....

### BEGIN /etc/grub.d/30_os-prober ###
### END /etc/grub.d/30_os-prober ###

### BEGIN /etc/grub.d/40_custom ###
### END /etc/grub.d/40_custom ###
.....（下面省略）.....
```

基本上，grub2 不希望你自己修改 grub.cfg 这个配置文件，取而代之的是修改几个特定的配置文件之后，由 grub2-mkconfig 这个指令来产生新的 grub.cfg 文件。 不过，你还是得要了解一下 grub2.cfg 的大致内容。

在 grub.cfg 最开始的部份，其实大多是环境设置与默认值设置等，比较重要的当然是默认由哪个选项开机 （set default） 以及默认的秒数 （set timeout）， 再来则是每一个菜单的设置，就是在“ menuentry ”这个设置值之后的项目啰！在鸟哥默认的配置文件当中，其实是有两个 menuentry 的， 也就是说，鸟哥的测试机在开机的时候应该就会有两个可以选择的菜单的意思啰！

在 menuentry 之后会有几个项目的规范，包括“ --class, --unrestricted --id ”等等的指定项目，之后通过“ { } ”将这个菜单会用到的数据框起来， 在选择这个菜单之后就会进行括号内的动作的意思。如果真的点选了这个菜单，那 grub2 首先会载入模块，例如上表中的“ load_video, insmod gzio, insmod part_gpt, insmod xfs ”等等的项目， 都是在载入要读取核心文件所需要的磁盘、分区、文件系统、解压缩等等的驱动程序。之后就是三个比较重要的项目：

- set root='hd0,gpt2' 这 root 是指定 grub2 配置文件所在的那个设备。以我们的测试机来说，当初安装的时候分区出 / 与 /boot 两个设备唷，而 grub2 是在 /boot/grub2 这个位置上， 而这个位置的磁盘文件名为 /dev/vda2 ，因此完整的 grub2 磁盘名称就是 （hd0,2） 啰！因为我们的系统用的是 GTP 的磁盘分区格式， 因此全名就是“ hd0,gpt2 ”！这样说，有没有听懂啊？

- linux16 /vmlinuz-... root=/dev/mapper/centos-root ... 这个就是 Linux 核心文件以及核心执行时所下达的参数。你应该会觉得比较怪的是，我们的核心文件不是 /boot/vmlinuz-xxx 吗？ 怎么这里的设置会是在根目录呢？这个跟上面的 root 有关啦！大部分的系统大多有 /boot 这个分区，如果 /boot 没有分区， 那会是怎么回事呢？我们用下面的叠代来说明一下：

    - 如果没有 /boot 分区，仅有 / 分区：所以文件名会这样变化喔： /boot/vmlinuz-xxx --> （/）/boot/vmlinuz-xxx --> （hd0,msdos1）/boot/vmlinuz-xxx
    - 如果 /boot 是独立分区，则文件名的变化会是这样： /boot/vmlinuz-xxx --> （/boot）/vmlinuz-xxx --> （hd0,msdos1）/vmlinuz-xxx因此，这个 linux16 后面接的文件名得要跟上面的 root 搭配在一起，才是完整的绝对路径文件名喔！看懂了吗？至于 linux16 /vmlinuz-xxx root=/file/name 那个 root 指的是“ linux 文件系统中，根目录是在哪个设备上”的意思！从本章一开始的开机流程中，我们就知道核心会主动去挂载根目录，并且从根目录中读取配置文件， 再进一步开始开机流程。所以，核心文件后面一定要接根目录的设备啊！这样理解吧？我们从 /etc/fstab 里面也知道根目录的挂载可以是设备文件名、 UUID 与 LABEL 名称，因此这个 root 后面也是可以带入类似 root=UUID=1111.2222.33... 之类的模式喔！

- initrd16 /initramfs-3.10... 这个就是 initramfs 所在的文件名，跟 linux16 那个 vmlinuz-xxx 相同，这个文件名也是需要搭配“ set root=xxx ”那个项目的设备， 才会得到正确的位置喔！注意注意！

19.3.3 grub2 配置文件维护 /etc/default/grub 与 /etc/grub.d

前一个小节我们谈到的是 grub2 的主配置文件 grub.cfg 约略的内容，但是因为该文件的内容太过复杂，数据量非常庞大，grub2 官方说明不建议我们手动修改！ 而是应该要通过 /etc/default/grub 这个主要环境配置文件与 /etc/grub.d/ 目录内的相关配置文件来处理比较妥当！ 我们先来聊聊 /etc/default/grub 这个主要环境配置文件好了！

- /etc/default/grub 主要环境配置文件

这个主配置文件的内容大概是长这样：

```
[root@study ~]# cat /etc/default/grub
GRUB_TIMEOUT=5                        # 指定默认倒数读秒的秒数
GRUB_DEFAULT=saved                    # 指定默认由哪一个菜单来开机，默认开机菜单之意
GRUB_DISABLE_SUBMENU=true             # 是否要隐藏次菜单，通常是藏起来的好！
GRUB_TERMINAL_OUTPUT="console"        # 指定数据输出的终端机格式，默认是通过文字终端机
GRUB_CMDLINE_LINUX="rd.lvm.lv=centos/root rd.lvm.lv=centos/swap crashkernel=auto rhgb qui
                                      # 就是在 menuentry 括号内的 linux16 项目后续的核心参数
GRUB_DISABLE_RECOVERY="true"          # 取消救援菜单的制作
```

有兴趣的伙伴请自行 info grub 并且找到 6.1 的章节阅读一下～我们下面主要谈的是几个重要的设置项目而已。现在来说说处理的项目重点吧！

- 倒数时间参数： GRUB_TIMEOUT

这个设置值相当简单，后面就是接你要倒数的秒数即可～例如要等待 30 秒，就在这边改成“GRUB_TIMEOUT=30”即可！如果不想等待则输入 0 ， 如果一定要使用者选择，则填 -1 即可！

- 是否隐藏菜单项目：GRUB_TIMEOUT_STYLE

这个项目可选择的设置值有 menu, countdown, hidden 等等。如果没有设置，默认是 menu 的意思。这个项目主要是在设置要不要显示菜单！ 如果你不想要让使用者看到菜单，这里可以设置为 countdown！那 countdown 与 hidden 有啥差异呢？countdown 会在屏幕上显示剩余的等待秒数， 而 hidden 则空空如也～除非你有特定的需求，否则这里一般鸟哥建议设置为 menu 较佳啦！

- 讯息输出的终端机模式：GRUB_TERMINAL_OUTPUT

这个项目是指定输出的画面应该使用哪一个终端机来显示的意思，主要的设置值有“ console, serial, gfxterm, vga_text ”等等。 除非有特别的需求，否则一般使用 console 即可！

- 默认开机菜单项目：GRUB_DEFAULT

这个项目在指定要用哪一个菜单 （menuentry） 来作为默认开机项目的意思。能使用的设置值包括有“ saved, 数字, title 名, ID 名”等等。 假设你有三笔 menuentry 的项目大约像这样：

```
menuentry '1st linux system' --id 1st-linux-system { ...}
menuentry '2nd linux system' --id 2nd-linux-system { ...}
menuentry '3rd win system' --id 3rd-win-system { ...}
```

几个常见的设置值是这样的：

```
[root@study ~]#
GRUB_DEFAULT=1
    代表使用第二个 menuentry 开机，因为数字的编号是以 0 号开始编的！

GRUB_DEFAULT=3rd-win-system
    代表使用第三个 menuentry 开机，因为里头代表的是 ID 的项目！它会找到 --id 喔！

GRUB_DEFAULT=saved
    代表使用 grub2-set-default 来设置哪一个 menuentry 为默认值的意思。通常默认为 0
```

一般来说，默认就是以第一个开机菜单来作为默认项目，如果想要有不同的菜单设置，可以在这个项目填选所需要的 --id 即可。 当然啦，你的 id 就应该不要重复啰！

- 核心的外加参数功能：GRUB_CMDLINE_LINUX

如果你的核心在启动的时候还需要加入额外的参数，就在这里加入吧！举例来说，如果你除了默认的核心参数之外，还需要让你的磁盘读写机制为 deadline 这个机制时， 可以这样处理：

```
GRUB_CMDLINE_LINUX="..... crashkernel=auto rhgb quiet elevator=deadline"
```

在暨有的项目之后加上如同上表的设置，这样就可以在开机时额外的加入磁盘读写的机制项目设置了！

这个主要环境配置文件编写完毕之后，必须要使用 grub2-mkconfig 来重建 grub.cfg 才行喔！因为主配置文件就是 grub.cfg 而已， 我们是通过许多脚本的协力来完成 grub.cfg 的自动创建。当然啰，额外自己设置的项目，就是写入 /etc/default/grub 文件内就是了。 我们来测试一下下面调整项目，看看你会不会修订主要环境配置文件了呢？

问：假设你需要（1）开机菜单等待 40 秒钟、（2）默认用第一个菜单开机、（3）菜单请显示出来不要隐藏、（4）核心外带“elevator=deadline”的参数值， 那应该要如何处理 grub.cfg 呢？答：直接编辑主要环境配置文件后，再以 grub2-mkconfig 来重建 grub.cfg 喔！

```
# 1\. 先编辑主要环境配置文件：
[root@study ~]# vim /etc/default/grub
GRUB_TIMEOUT=40
GRUB_DEFAULT=0
GRUB_TIMEOUT_STYLE=menu
GRUB_DISABLE_SUBMENU=true
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="rd.lvm.lv=centos/root rd.lvm.lv=centos/swap crashkernel=auto rhgb
  quiet elevator=deadline"
GRUB_DISABLE_RECOVERY="true"

# 2\. 开始重新创建 grub.cfg ！
[root@study ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.10.0-229.el7.x86_64
Found initrd image: /boot/initramfs-3.10.0-229.el7.x86_64.img
Found linux image: /boot/vmlinuz-0-rescue-309eb890d09f440681f596543d95ec7a
Found initrd image: /boot/initramfs-0-rescue-309eb890d09f440681f596543d95ec7a.img
done

# 3\. 检查看看 grub.cfg 的内容是否真的是改变了？
[root@study ~]# grep timeout /boot/grub2/grub.cfg
  set timeout_style=menu
  set timeout=40

[root@study ~]# grep default /boot/grub2/grub.cfg
   set default="0"

[root@study ~]# grep linux16 /boot/grub2/grub.cfg
        linux16 /vmlinuz-3.10.0-229.el7.x86_64 root=/dev/.... elevator=deadline
        linux16 /vmlinuz-0-rescue-309eb890d09f440681f5965.... elevator=deadline
```

- 菜单创建的脚本 /etc/grub.d/*

你应该会觉得很奇怪， grub2-mkconfig 执行之后，屏幕怎么会主动的去抓到 linux 的核心，还能够找到对应核心版本的 initramfs 呢？ 怎么这么厉害？其实 grub2-mkconfig 会去分析 /etc/grub.d/ 里面的文件，然后执行该文件来创建 *grub.cfg* 的啦！ 所以啰， */etc/grub.d/* 里面的文件就显得很重要了。一般来说，该目录下会有这些文件存在：

- 00_header：主要在创建初始的显示项目，包括需要载入的模块分析、屏幕终端机的格式、倒数秒数、菜单是否需要隐藏等等，大部分在 /etc/default/grub 里面所设置的变量，大概都会在这个脚本当中被利用来重建 grub.cfg 。

- 10_linux：根据分析 /boot 下面的文件，尝试找到正确的 linux 核心与读取这个核心需要的文件系统模块与参数等，都在这个脚本运行后找到并设置到 grub.cfg 当中。 因为这个脚本会将所有在 /boot 下面的每一个核心文件都对应到一个菜单，因此核心文件数量越多，你的开机菜单项目就越多了。 如果未来你不想要旧的核心出现在菜单上，那可以通过移除旧核心来处理即可。

- 30_os-prober：这个脚本默认会到系统上找其他的 partition 里面可能含有的操作系统，然后将该操作系统做成菜单来处理就是了。 如果你不想要让其他的操作系统被侦测到并拿来开机，那可以在 /etc/default/grub 里面加上“ GRUB_DISABLE_OS_PROBER=true ”取消这个文件的运行。

- 40_custom：如果你还有其他想要自己手动加上去的菜单项目，或者是其他的需求，那么建议在这里补充即可！

所以，一般来说，我们会更动到的就是仅有 40_custom 这个文件即可。那这个文件内容也大多在放置管理员自己想要加进来的菜单项目就是了。 好了，那问题来了，我们知道 menuentry 就是一个菜单，那后续的项目有哪些东西呢？简单的说，就是这个 menuentry 有几种常见的设置？ 亦即是 menuentry 的功能啦！常见的有这几样：

- 直接指定核心开机

基本上如果是 Linux 的核心要直接被用来开机，那么你应该要通过 grub2-mkconfig 去抓 10_linux 这个脚本直接制作即可，因此这个部份你不太需要记忆！ 因为在 grub.cfg 当中就已经是系统能够捉到的正确的核心开机菜单了！不过如果你有比较特别的参数需要进行呢？这时候你可以这样作：（1）先到 grub.cfg 当中取得你要制作的那个核心的菜单项目，然后将它复制到 40_custom 当中 （2）再到 40_custom 当中依据你的需求修改即可。

这么说或许你很纳闷，我们来做个实际练习好了：

问：如果你想要使用第一个原有的 menuentry 取出来后，增加一个菜单，该菜单可以强制 systemd 使用 graphical.target 来启动 Linux 系统， 让该菜单一定可以使用图形界面而不用理会 default.target 的链接，该如何设计？答：当核心外带参数中，有个“ systemd.unit=??? ”的外带参数可以指定特定的 target 开机！因此我们先到 grub.cfg 当中，去复制第一个 menuentry ， 然后进行如下的设置：

```
[root@study ~]# vim /etc/grub.d/40_custom
menuentry 'My graphical CentOS, with Linux 3.10.0-229.el7.x86_64' --class rhel fedora
          --class gnu-linux --class gnu --class os --unrestricted --id 'mygraphical' {
        load_video
        set gfxpayload=keep
        insmod gzio
        insmod part_gpt
        insmod xfs
        set root='hd0,gpt2'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint='hd0,gpt2'  94ac5f77-cb8a-495e-a
        else
          search --no-floppy --fs-uuid --set=root 94ac5f77-cb8a-495e-a65b-2ef7442b837c
        fi
        linux16 /vmlinuz-3.10.0-229.el7.x86_64 root=/dev/mapper/centos-root ro rd.lvm.lv=
                centos/root rd.lvm.lv=centos/swap crashkernel=auto rhgb quiet
                elevator=deadline systemd.unit=graphical.target
        initrd16 /initramfs-3.10.0-229.el7.x86_64.img
}
# 请注意，上面的数据都是从 grub.cfg 里面复制过来的，增加的项目仅有特殊字体的部份而已！
# 同时考虑画面宽度，该项目稍微被变动过，请依据您的环境来设置喔！

[root@study ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

当你再次 reboot 时，系统就会多出一个菜单给你选择了！而且选择该菜单之后，你的系统就可以直接进入图形界面（如果有安装相关的 X window 软件时）， 而不必考虑 default.target 是啥东西了！了解乎？

- 通过 chainloader 的方式移交 loader 控制权

所谓的 chain loader （开机管理程序的链结） 仅是在将控制权交给下一个 boot loader 而已，所以 grub2 并不需要认识与找出 kernel 的文件名 ，“ 他只是将 boot 的控制权交给下一个 boot sector 或 MBR 内的 boot loader 而已 ” 所以通常他也不需要去查验下一个 boot loader 的文件系统！

一般来说， chain loader 的设置只要两个就够了，一个是预计要前往的 boot sector 所在的分区代号， 另一个则是设置 chainloader 在那个分区的 boot sector （第一个扇区） 上！假设我的 Windows 分区在 /dev/sda1 ，且我又只有一颗硬盘，那么要 grub 将控制权交给 windows 的 loader 只要这样就够了：

```
menuentry "Windows" {
        insmod chain      # 你得要先载入 chainloader 的模块对吧？
        insmod ntfs       # 建议加入 windows 所在的文件系统模块较佳！
        set root=（hd0,1）  # 是在哪一个分区～最重要的项目！
        chainloader +1    # 请去 boot sector 将 loader 软件读出来的意思！
}
```

通过这个项目我们就可以让 grub2 交出控制权了！

问：假设你的测试系统上面使用 MBR 分区，并且出现如下的数据：

```
[root@study ~]# fdisk -l /dev/vda
   Device Boot      Start         End      Blocks   Id  System
/dev/vda1            2048    10487807     5242880   83  Linux
/dev/vda2   *    10487808   178259967    83886080    7  HPFS/NTFS/exFAT
/dev/vda3       178259968   241174527    31457280   83  Linux
```

其中 /dev/vda2 使用是 windows 7 的操作系统。现在我需要增加两个开机选项，一个是取得 windows 7 的开机菜单，一个是回到 MBR 的默认环境，应该如何处理呢？答：windows 7 在 /dev/vda2 亦即是 hd0,msdos2 这个地方，而 MBR 则是 hd0 即可，不需要加上分区啊！因此整个设置会变这样：

```
[root@study ~]# vim /etc/grub.d/40_custom
menuentry 'Go to Windows 7' --id 'win7' {
        insmod chain
        insmod ntfs
        set root=（hd0,msdos2）
        chainloader +1
}
menuentry 'Go to MBR' --id 'mbr' {
        insmod chain
        set root=（hd0）
        chainloader +1
}

[root@study ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

另外，如果每次都想要让 windows 变成默认的开机选项，那么在 /etc/default/grub 当中设置好“ GRUB_DEFAULT=win7 ” 然后再次 grub2-mkconfig 这样即可啦！不要去算 menuentry 的顺序喔！通过 --id 内容来处理即可！

19.3.4 initramfs 的重要性与创建新 initramfs 文件

我们在本章稍早之前“ boot loader 与 kernel 载入”的地方已经提到过 initramfs 这玩意儿，他的目的在于提供开机过程中所需要的最重要核心模块，以让系统开机过程可以顺利完成。 会需要 initramfs 的原因，是因为核心模块放置于 /lib/modules/$（uname -r）/kernel/ 当中， 这些模块必须要根目录 （/） 被挂载时才能够被读取。但是如果核心本身不具备磁盘的驱动程序时， 当然无法挂载根目录，也就没有办法取得驱动程序，因此造成两难的地步。

initramfs 可以将 /lib/modules/.... 内的“开机过程当中一定需要的模块”包成一个文件 （文件名就是 initramfs）， 然后在开机时通过主机的 INT 13 硬件功能将该文件读出来解压缩，并且 initramfs 在内存内会仿真成为根目录， 由于此虚拟文件系统 （Initial RAM Disk） 主要包含磁盘与文件系统的模块，因此我们的核心最后就能够认识实际的磁盘， 那就能够进行实际根目录的挂载啦！所以说：“initramfs 内所包含的模块大多是与开机过程有关，而主要以文件系统及硬盘模块 （如 usb, SCSI 等） 为主”的啦！

一般来说，需要 initramfs 的时刻为：

- 根目录所在磁盘为 SATA、USB 或 SCSI 等连接接口；
- 根目录所在文件系统为 LVM, RAID 等特殊格式；
- 根目录所在文件系统为非传统 Linux 认识的文件系统时；
- 其他必须要在核心载入时提供的模块。



Tips 之前鸟哥忽略 initrd 这个文件的重要性，是因为鸟哥很穷... ^_^。因为鸟哥的 Linux 主机都是较早期的硬件， 使用的是 IDE 接口的硬盘，而且并没有使用 LVM 等特殊格式的文件系统，而 Linux 核心本身就认识 IDE 接口的磁盘， 因此不需要 initramfs 也可以顺利开机完成的。自从 SATA 硬盘流行起来后，没有 initramfs 就没办法开机了！ 因为 SATA 硬盘使用的是 SCSI 模块来驱动的，而 Linux 默认将 SCSI 功能编译成为模块....

一般来说，各 distribution 提供的核心都会附上 initramfs 文件，但如果你有特殊需要所以想重制 initramfs 文件的话， 可以使用 dracut / mkinitrd 来处理的。这个文件的处理方式很简单， man dracut 或 man mkinitrd 就知道了！ ^_^。 CentOS 7 应该要使用 dracut 才对，不过 mkinitrd 还是有保留下来，两者随便你玩！鸟哥这里主要是介绍 dracut 就是了！

```
[root@study ~]# dracut [-fv] [--add-drivers 列表] initramfs文件名 核心版本
选项与参数：
-f    ：强迫编译出 initramfs ，如果 initramfs 文件已经存在，则覆盖掉旧文件
-f    ：显示 dracut 的运行过程
--add-drivers 列表：在原本的默认核心模块中，增加某些你想要的模块！模块位于核心所在目录
                    /lib/modules/$（uname -r）/kernel/*
initramfs文件名      ：就是你需要的文件名！开头最好就是 initramfs，后面接版本与功能
核心版本           ：默认当然是目前运行中的核心版本，不过你也可以手动输入其他不同版本！
其实 dracut 还有很多功能，例如下面的几个参数也可以参考看看：
--modules   ：将 dracut 所提供的开机所需模块 （核心核模块） 载入，可用模块在下面的目录内
             /usr/lib/dracut/modules.d/
--gzip&#124;--bzip2&#124;--xz：尝试使用哪一种压缩方式来进行 initramfs 压缩。默认使用 gzip 喔！
--filesystems ：加入某些额外的文件系统支持！

范例一：以 dracut 的默认功能创建一个 initramfs 虚拟磁盘文件
[root@study ~]# dracut -v initramfs-test.img $（uname -r）
Executing: /sbin/dracut -v initramfs-test.img 3.10.0-229.el7.x86_64
*** Including module: bash ***                      # 先载入 dracut 本身的模块支持
*** Including module: nss-softokn ***
*** Including modules done ***
.....（中间省略）..... # 下面两行在处理核心模块
*** Installing kernel module dependencies and firmware ***
*** Installing kernel module dependencies and firmware done ***
.....（中间省略）.....
*** Generating early-microcode cpio image ***      # 创建微指令集
*** Constructing GenuineIntel.bin ****
*** Store current command line parameters ***
*** Creating image file ***                        # 开始创建 initramfs 啰！
*** Creating image file done ***

范例二：额外加入 e1000e 网卡驱动与 ext4/nfs 文件系统在新的 initramfs 内
[root@study ~]# dracut -v --add-drivers "e1000e" --filesystems "ext4 nfs" \
&gt;  initramfs-new.img $（uname -r）
[root@study ~]# lsinitrd initramfs-new.img  &#124; grep -E '（e1000&#124;ext4&#124;nfs）'
 usr/lib/modules/3.10.0-229.el7.x86_64/kernel/drivers/net/ethernet/intel/e1000e
 usr/lib/modules/3.10.0-229.el7.x86_64/kernel/drivers/net/ethernet/intel/e1000e/e1000e.ko
 usr/lib/modules/3.10.0-229.el7.x86_64/kernel/fs/ext4
 usr/lib/modules/3.10.0-229.el7.x86_64/kernel/fs/ext4/ext4.ko
 usr/lib/modules/3.10.0-229.el7.x86_64/kernel/fs/nfs
 usr/lib/modules/3.10.0-229.el7.x86_64/kernel/fs/nfs/nfs.ko
# 你可以看得到，新增的模块现在正在新的 initramfs 当中了呢！很愉快喔！
```

initramfs 创建完成之后，同时核心也处理完毕后，我们就可以使用 grub2 来创建菜单了！下面继续瞧一瞧吧！

19.3.5 测试与安装 grub2

如果你的 Linux 主机本来就是使用 grub2 作为 loader 的话，那么你就不需要重新安装 grub2 了， 因为 grub2 本来就会主动去读取配置文件啊！您说是吧！但如果你的 Linux 原来使用的并非 grub2 ， 那么就需要来安装啦！如何安装呢？首先，你必须要使用 grub-install 将一些必要的文件复制到 /boot/grub2 里面去，你应该这样做的：

```
[root@study ~]# grub2-install [--boot-directory=DIR] INSTALL_DEVICE
选项与参数：
--boot-directory=DIR 那个 DIR 为实际的目录，使用 grub2-install 默认会将
  grub2 所有的文件都复制到 /boot/grub2/* ，如果想要复制到其他目录与设备去，
  就得要用这个参数。
INSTALL_DEVICE 安装的设备代号啦！

范例一：将 grub2 安装在目前系统的 MBR 下面，我的系统为 /dev/vda：
[root@study ~]# grub2-install /dev/vda
# 因为原本 /dev/vda 就是使用 grub2 ，所以似乎不会出现什么特别的讯息。
# 如果去查阅一下 /boot/grub2 的内容，会发现所有的文件都更新了，因为我们重装了！
# 但是注意到，我们并没有配置文件喔！那要自己创建！
```

基本上，grub2-install 大概仅能安装 grub2 主程序与相关软件到 /boot/grub2/ 那个目录去，如果后面的设备填的是整个系统 （/dev/vda, /dev/sda...），那 loader 的程序才会写入到 MBR 里面去。如果是 XFS 文件系统的 /dev/vda2 设备的话 （个别 partition），那 grub2-install 就会告诉你， 该文件系统并不支持 grub2 的安装喔！也就是你不能用 grub2-install 将你的主程序写入到 boot sector 里头去的意思啦！ 那怎办？没关系，来强迫写入一下看看！

```
# 尝试看一下你的系统中有没有其他的 xfs 文件系统，且为传统的 partition 类型？
[root@study ~]# df -T &#124;grep -i xfs
/dev/mapper/centos-root   xfs       10475520 4128728   6346792  40% /
/dev/mapper/centos-home   xfs        5232640  665544   4567096  13% /home
/dev/mapper/raidvg-raidlv xfs        1558528   33056   1525472   3% /srv/raidlvm
/dev/vda2                 xfs        1038336  144152    894184  14% /boot
/dev/vda4                 xfs        1038336   63088    975248   7% /srv/myproject
# 看起来仅有 /dev/vda4 比较适合做个练习的模样了！来瞧瞧先！

# 将 grub2 的主程序安装到 /dev/vda4 去看看！
[root@study ~]# grub2-install /dev/vda4
Installing for i386-pc platform.
grub2-install: error: hostdisk//dev/vda appears to contain a xfs filesystem which isn't
  known to reserve space for DOS-style boot.  Installing GRUB there could result in
  FILESYSTEM DESTRUCTION if valuable data is overwritten by grub-setup （--skip-fs-probe
  disables this check, use at your own risk）.
# 说是 xfs 恐怕不能支持你的 boot sector 概念！这个应该是误判！所以我们还是给它强制装一下！

[root@study ~]# grub2-install --skip-fs-probe /dev/vda4
Installing for i386-pc platform.
grub2-install: warning: File system ‘xfs’ doesn't support embedding.
grub2-install: warning: Embedding is not possible.  GRUB can only be installed in this
  setup by using blocklists.  However, blocklists are UNRELIABLE and their use is
  discouraged..
grub2-install: error: will not proceed with blocklists.
# 还是失败！因为还是担心 xfs 被搞死～好！没问题！加个 --force 与 --recheck 重新处理一遍！

[root@study ~]# grub2-install --force --recheck --skip-fs-probe /dev/vda4
Installing for i386-pc platform.
grub2-install: warning: File system ‘xfs’ doesn't support embedding.
grub2-install: warning: Embedding is not possible.  GRUB can only be installed in this
  setup by using blocklists.  However, blocklists are UNRELIABLE and their use is
  discouraged..
Installation finished. No error reported.
# 注意看！原本是无法安装的错误，现在仅有 warning 警告讯息，所以这样就安装到 partition 上了！
```

上面这样就将 grub2 的主程序安装到 /dev/vda4 以及重新安装到 MBR 里面去了。现在来思考一下，我们知道 grub2 主程序会去找 grub.cfg 这个文件，大多是在 /boot/grub2/grub.cfg 里面，那有趣了，我们的 MBR 与 /dev/vda4 都是到 /boot/grub2/grub.cfg 去抓设置吗？ 如果是多重操作系统那怎办？呵呵！这就需要重新进入新系统才能够安装啦！举个例子来说啰：

问：假设你的测试系统上面使用 MBR 分区，并且出现如下的数据：

```
[root@study ~]# fdisk -l /dev/vda
   Device Boot      Start         End      Blocks   Id  System
/dev/vda1            2048    10487807     5242880   83  Linux
/dev/vda2   *    10487808   178259967    83886080    7  HPFS/NTFS/exFAT
/dev/vda3       178259968   241174527    31457280   83  Linux
```

其中 /dev/vda1, /dev/vda3 是两个 CentOS 7 系统，而 /dev/vda2 则是 windows 7 系统。安装的流程是依序 /dev/vda1 --> /dev/vda2 --> /dev/vda3。因此，安装好而且重新开机后，系统其实是默认进入 /dev/vda3 这个 CentOS 7 的系统的。 此时 MBR 会去读取的配置文件在（/dev/vda3）/boot/grub2/grub.cfg 才对。

因为 /dev/vda1 应该是用来管理开机菜单的，而 /dev/vda2 及 /dev/vda3 在规划中就是用来让学生操作的，因此默认情况下， /dev/vda1 内的 CentOS 系统应该只会在开机的时候用到而已，或者是出问题时会找他来使用。至于 /dev/vda3 及 /dev/vda2 则可能因为学生的误用，因此未来可能会升级或删除或重灌等。那你如何让系统永远都是使用 /dev/vda1 开机呢？答：因为 MBR 的 boot loader 应该要去 （/dev/vda1）/boot/grub2/grub.cfg 读取相关设置才是正常的！所以，你可以使用几种基本的方式来处理：

- 因为 CentOS 7 会主动找到其他操作系统，因此你可以在 /dev/vda3 的开机菜单中找到 /dev/vda1 的开机选项，请用该选项进入系统， 你就能够进入 /dev/vda1 了！
- 假设没能抓到 /dev/vda1 ，那你可以在 /dev/vda3 下面使用 chroot 来进入 /dev/vda1 喔！
- 使用救援光盘去抓到正确的 /dev/vda1，然后取得 /dev/vda1 的系统喔！

等到进入系统后，修改 /etc/default/grub 及 /etc/grub.d/40_custom 之后，使用 grub2-mkconfig -o /boot/grub2/grub.cfg ， 然后重新 grub2-install /dev/vda 就能够让你的 MBR 去取得 /dev/vda1 内的配置文件啰！

问：依据 19.3.3 小节的第一个练习，我们的测试机目前为 40 秒倒数，且有一个强制进入图形界面的“ My graphical CentOS7 ”菜单！ 现在我们想要多加两个菜单，一个是回到 MBR 的 chainloader，一个是使用 /dev/vda4 的 chainloader，该如何处理？答：因为没有必要重新安装 grub2 ，直接修改即可。修改 40_custom 成为这样：

```
[root@study ~]# vim /etc/grub.d/40_custom
# 最下面加入这两个项目即可！
menuentry 'Goto MBR' {
        insmod chain
        insmod part_gpt
        set root=（hd0）
        chainloader +1
}
menuentry 'Goto /dev/vda4' {
        insmod chain
        insmod part_gpt
        set root=（hd0.gpt4）
        chainloader +1
}

[root@study ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

最后总结一下：

1. 如果是从其他 boot loader 转成 grub2 时，得先使用 grub2-install 安装 grub2 配置文件；
2. 承上，如果安装到 partition 时，可能需要加上额外的许多参数才能够顺利安装上去！
3. 开始编辑 /etc/default/grub 及 /etc/grub.d/* 这几个重要的配置文件；
4. 使用 grub2-mkconfig -o /boot/grub2/grub.cfg 来创建开机的配置文件！

19.3.6 开机前的额外功能修改

事实上，前几个小节设置好之后，你的 grub2 就已经在你的 Linux 系统上面了，而且同时存在于 MBR 与 boot sector 当中呢！所以，我们已经可以重新开机来查阅看看啦！ 另外，如果你正在进行开机，那么请注意，我们可以在默认菜单 （鸟哥的范例当中是 40 秒） 按下任意键， 还可以进行 grub2 的"线上编修"功能喔！真是棒啊！先来看看开机画面吧！

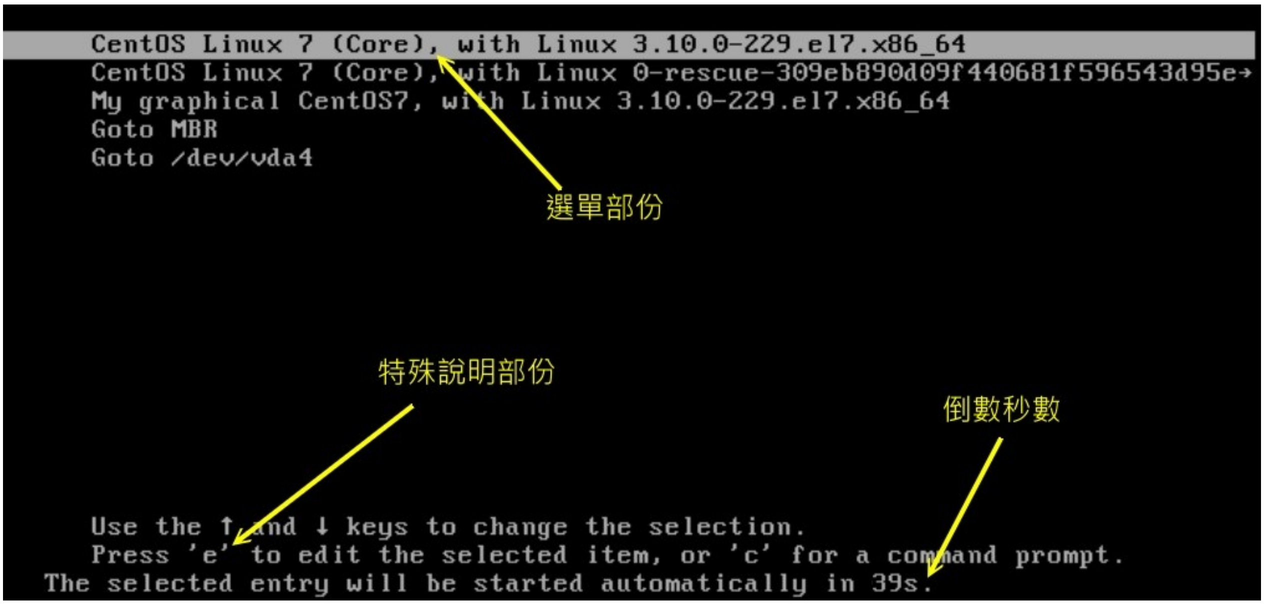


图19.3.1、grub2 开机画面示意图

由于默认菜单就没有隐藏，因此你会直接看到这 5 个菜单而已，同时会有读秒的咚咚在倒数。 菜单部分的画面其实就是 menuentry 后面的文字啦！你现在知道如何修改 menuentry 后面的文字了吧！ ^_^。 然后如果你点选了“Goto MBR”与“Goto /dev/vda4”时，怪了！怎么发现到菜单又重新回来了呢？ 这是因为这两个 Goto 的菜单都是重新读取主配置文件，而 MBR 与 /dev/vda4 配置文件的读取都是来自 （/dev/vda2）/boot/grub2/grub.cfg 的缘故！ 因此这个画面就会重复出现了！这样了解乎？

另外，如果你再仔细看的话，会发现到上图中底部还有一些细部的选项，似乎有个 'e' edit 的样子！ 没错～ grub2 支持线上编修指令喔！这是个很有用的功能！假如刚刚你将 grub.cfg 的内容写错了，导致出现无法开机的问题时， 我们可以查阅该 menuentry 菜单的内容并加以修改喔！举例来说，我想要知道第一个菜单的实际内容时，将反白光棒移动到第一个菜单， 再按下 'e' 会进入如下画面：

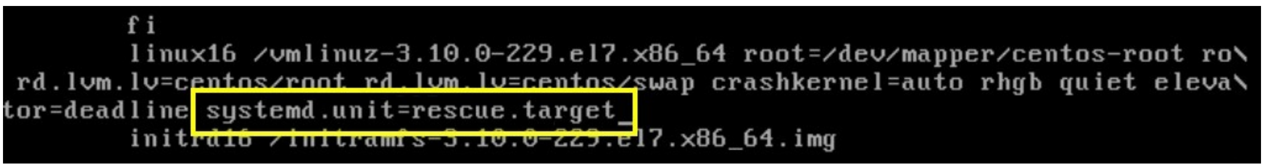


图19.3.2、grub2 额外的指令编辑模式

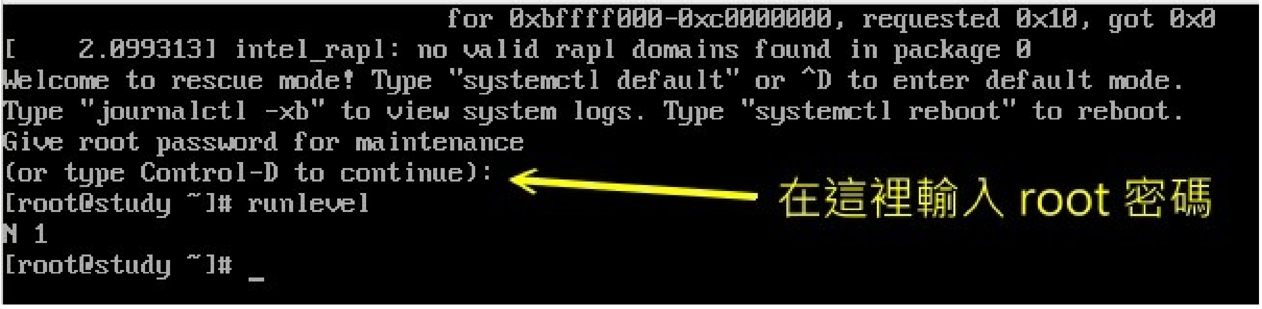
因为 CentOS 7 默认没有提供美美的底图给我们使用，因此这里会看到无法分辨的两个区块！事实上它真的是两个区块， 上方是实际你可以编辑的内容区段，仔细看，这不就是我们在 grub.cfg 里面设置的东西吗？没错！此时你还可以继续进一步修改喔！ 用上/下/左/右按键到你想要编辑的地方，直接删除、新增即可！

至于下方画面则仅是一些编辑说明，重点在告诉你，编辑完毕之后，若想要取消而回到前一个画面，请使用 [crtl]+c 或者是 [esc] 回去， 若是修改完毕，想要直接开机时，请使用 [crtl]+x 来开机啰！

问：现在我想要让系统开机的过程中，让这个系统进入救援模式 （rescue） ，而不想要进入系统后使用 systemctl rescue 时，该如何处理？答：仔细看到图 19.3.2 的画面，按下“向下”的方向键，直到出现 linux16 那一行，然后在那一行的最后面加上 systemd.unit=rescue.target ， 画面有点像这样：



然后再按下 [crtl]+x 来进入系统，就能够取得 rescue 的环境了！登陆后有点像这样：



接着下来你就可以开始救援系统啰！

你可能会觉得很讶异！早期 SystemV 的系统中，进入 runlevel 1 的状态是不需要输入 root 密码的，在 systemd 的年代，哇！！ 竟然需要密码才能够进入救援模式耶！而且是强制要有 root 密码耶！如果你是 root 密码忘记要救援，救个鬼啊～还是需要 root 密码啊！ 那怎办？没关系～本章稍后会告诉你应该要如何处理的啦！

19.3.7 关于开机画面与终端机画面的图形显示方式

如果你想要让你的开机画面使用图形显示方式，例如使用中文来显示你的画面啊！因为我们默认的 locale 语系就是 zh_TW.utf8 嘛！ 所以理论上 grub2 会显是中文出来才对啊！有没有办法达成呢？是有的～通过图形显是的方法即可！不过， 我们得要重新修改 grub.cfg 才行喔！依据下面的方式来处理：

```
# 先改重要的配置文件
[root@study ~]# vim /etc/default/grub
.....（前面省略）.....
GRUB_TERMINAL=gfxterm        # 设置主要的终端机显示为图形界面！
GRUB_GFXMODE=1024x768x24     # 图形界面的 X, Y, 彩度数据
GRUB_GFXPAYLOAD_LINUX=keep   # 保留图形界面，不要使用 text 喔！

# 重新创建配置文件
[root@study ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

再次的重新开机，这时你会看到有点像下面的模样的画面喔！

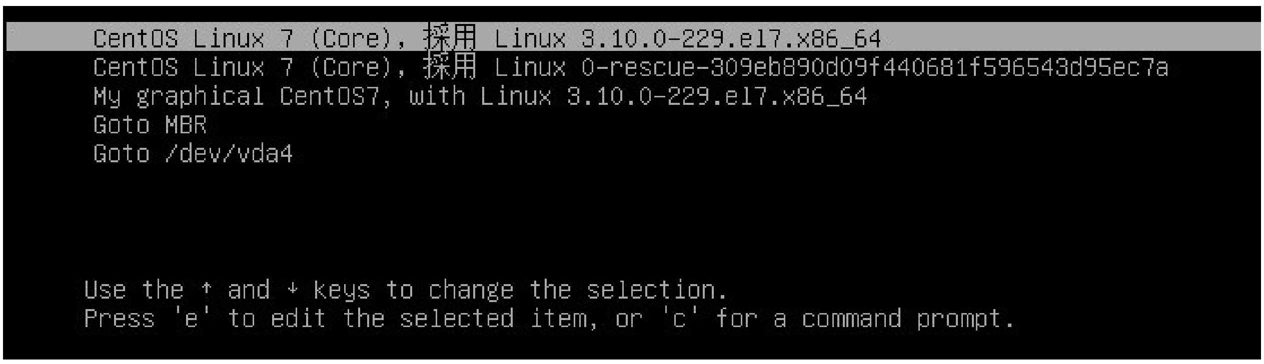


图19.3.3、使用图形显示模式的开机画面

看到没有？上图中有繁体中文喔！中文喔喔喔喔喔喔～真是开心啊！ 未来如果你有需要在你的开机菜单当中加入许多属于你自己的公司/企业的画面，那就太容易啰！ ^_^

19.3.8 为个别菜单加上密码

想像一个环境，如果你管理的是一间计算机教室，这间计算机教室因为可对外开放，但是你又担心某些 partition 被学生不小心的弄乱，因此你可能会想要将某些开机菜单作个保护。这个时候，为每个菜单作个加密的密码就是个可行的方案啦！

另外，从本章前面的 19.3.6 小节介绍的开机过程中，你会知道使用者可以在开机的过程中于 grub2 内选择进入某个菜单，以及进入 grub2 指令模式去修改菜单的参数数据等。 也就是说，主要的 grub2 控制有：（1）grub2 的菜单命令行修改与（2）进入选择的菜单开机流程。好了，如刚刚谈到的计算机教室案例， 你要怎么让某些密码可以完整的掌控 grub2 的所有功能，某些密码则只能进入个别的菜单开机呢？这就得要牵涉到 grub2 的帐号机制了！

- grub2 的帐号、密码与菜单设置

grub2 有点在仿真 Linux 的帐号管理方案喔！因为在 grub2 的菜单管理中，有针对两种身份进行密码设置：

- superusers： 设置系统管理员与相关参数还有密码等，使用这个密码的用户，将可在 grub2 内具有所有修改的权限。 但一旦设置了这个 superusers 的参数，则所有的指令修改将会被变成受限制的！
- users： 设置一般帐号的相关参数与密码，可以设置多个用户喔！使用这个密码的用户可以选择要进入某些菜单项目。 不过，菜单项目也得要搭配相对的帐号才行喔！（一般来说，使用这种密码的帐号并不能修改菜单的内容，仅能选择进入菜单去开机而已）

这样说可能你不是很容易看得懂，我们使用下面的一个范例来说明你就知道怎么处理了。另外，下面的范例是单纯给读者们看看而已的～ 不能够直接用在我们的测试机器里面喔！

问：假设你的系统有三个各别的操作系统，分别安装在（hd0,1），（hd0,2），（hd0,3）当中。假设（hd0,1）是所有人都可以选择进入的系统，（hd0,2）是只有系统管理员可以进入的系统，（hd0,3）则是另一个一般用户与系统管理员可以进入的系统。另外，假设系统管理员的帐号/密码设置为 vbird/abcd1234， 而一般帐号为 dmtsai/dcba4321 ，那该如何设置？
答：如果依据上述的说明，其实没有用到 Linux 的 linux16 与 initrd16 的项目，只需要 chainloader 的项目而已！ 因此，整个 grub.cfg 会有点像下面这样喔：

```
# 第一个部份是先设置好管理员与一般帐号的帐号名称与密码项目！
 set superusers="vbird"    # 这里是设置系统管理员的帐号名称为啥的意思！
password vbird abcd1234   # 当然要给予这个帐号密码啊！
password dmtsai dcba4321  # 没有输入 superuses 的其他帐号，当然就是判定为一般帐号

menuentry "大家都可以选择我来开机喔！" --unrestricted {
    set root=(hd0,1)
    chainloader +1
}

menuentry "只有管理员的密码才有办法使用" --users "" {
    set root=(hd0,2)
    chainloader +1
}

menuentry "只有管理员与 dmtsai 才有办法使用喔！" --users dmtsai {
    set root=(hd0,3)
    chainloader +1
}
```

如上表所示，你得要使用 superuses 来指定哪个帐号是管理员！另外，这个帐号与 Linux 的实体帐号无关，这仅是用来判断密码所代表的意义而已。 而密码的给予有两种语法：

- password_pbkdf2 帐号 “使用grub2-mkpasswd-pbkdf2所产生的密码”
- password 帐号 “没加密的明码”

有了帐号与密码之后，在来就是在个别的菜单上面加上是否要取消限制 （--unrestricted） 或者是给予哪个用户 （--users） 的设置项目。 同时请注意喔，所有的系统管理员所属的密码应该是能够修改所有的菜单，因此你无须在第三个菜单上面加入 vbird 这个管理员帐号！ 这样说你就可以了解了吧？

你很可能会这样说：“了解个头啦！怎么可能会了解！前面不是才说过：「不要手动去修改 grub.cfg 」吗？这里怎么直接列出 grub.cfg 的内容？ 上面这些项目我是要在哪些环境配置文件里面修改啦？”呵呵～您眞内行，没有被骗耶～好厉害～好厉害！

- grub2 密码设置的文件位置与加密的密码

还记得我们在前几小节谈到主要的环境设置是在 /etc/grub.d/* 里面吧？里面的文件文件名有用数字开头，那些数字照顺序，就是 grub.cfg 的来源顺序了。 因此最早被读的应该是 00_header，但是那个文件的内容挺重要的，所以 CentOS 7 不建议你改它～那要改谁？就自己创建一个名为 01_users 的文件即可！ 要注意是两个数字开头接着底线的文件名才行喔！然后将帐号与密码参数给它补进去！

现在让我们将 vbird 与 dmtsai 的密码加密，实际在我们的测试机器上面创建起来吧！

```
# 1\. 先取得 vbird 与 dmtsai 的密码。下面我仅以 vbird 来说明而已！
[root@study ~]# grub2-mkpasswd-pbkdf2
Enter password:    # 这里输入你的密码
Reenter password:  # 再一次输入密码
PBKDF2 hash of your password is grub.pbkdf2.sha512.10000.9A2EBF7A1F484...
# 上面特殊字体从 grub.pbkdf2.... 的那一行，全部的数据就是你的密码喔！复制下来！

# 2\. 将密码与帐号写入到 01_users 文件内
[root@study ~]# vim /etc/grub.d/01_users
cat &lt;&lt; eof
set superusers="vbird"
password_pbkdf2 vbird grub.pbkdf2.sha512.10000.9A2EBF7A1F484904FF3681F97AE22D58DFBFE65A..
password_pbkdf2 dmtsai grub.pbkdf2.sha512.10000.B59584C33BC12F3C9DB8B18BE9F557631473AED..
eof
# 请特别注意，在 /etc/grub.d/* 下面的文件是“执行脚本”档，是要被执行的！
# 因此不能直接写帐密，而是通过 cat 或 echo 等指令方式来将帐密数据显示出来才行喔！

# 3\. 因为 /etc/grub.d/ 下面应该是可执行文件，所以刚刚创建的 01_users 当然要给予执行权限
[root@study ~]# chmod a+x /etc/grub.d/01_users
[root@study ~]# ll /etc/grub.d/01_users
-rwxr-xr-x. 1 root root 649 Aug 31 19:42 /etc/grub.d/01_users
```

很快的，你就已经将密码创建妥当了！接下来就来聊一聊，那么每个 menuentry 要如何修改呢？

- 为个别的菜单设置帐号密码的使用模式

回想一下我们之前的设置，目前测试机器的 Linux 系统菜单应该有五个：

- 来自 /etc/grub.d/10_linux 这个文件主动侦测的两个 menuentry；
- 来自 /etc/grub.d/40_custom 这个我们自己设置的三个 menuentry

在 40_custom 内的设置，我们可以针对每个 menuentry 去调整，而且该调整是固定的，不会随便被更改。至于 10_linux 文件中， 则每个 menuentry 的设置都会依据 10_linux 的数据去变更，也就是由 10_linux 侦测到的核心开机菜单都会是相同的意思。

因为我们已经在 01_users 文件内设置了 set superusers="vbird" 这个设置值，因此每个菜单内的参数除了知道 vbird 密码的人之外， 已经不能随便修改了喔！所以，选择 10_linux 制作出来的菜单开机，应该就算正常开机，所以，我们默认不要使用密码好了！ 刚刚好 10_linux 的 menuentry 设置值就是这样：

```
[root@study ~]# vim /etc/grub.d/10_linux
.....（前面省略）.....
CLASS="--class gnu-linux --class gnu --class os --unrestricted"
# 这一行大约在 29 行左右，你可以利用 unrestricted 去搜寻即可！
# 默认已经不受限制 （--unrestricted） 了！如果想要受限制，在这里将 --unrestricted
# 改成你要使用的 --users "帐号名称" 即可！不过，还是不建议修改啦！
```

现在我们假设在 40_custom 里面要增加一个可以进入救援模式 （rescue） 的环境，并且放置到最后一个菜单中，同时仅有知道 dmtsai 的密码者才能够使用， 那你应该这样作：

```
[root@study ~]# vim /etc/grub.d/40_custom
.....（前面省略）.....
menuentry 'Rescue CentOS7, with Linux 3.10.0-229.el7.x86_64' --users dmtsai {
        load_video
        set gfxpayload=keep
        insmod gzio
        insmod part_gpt
        insmod xfs
        set root='hd0,gpt2'
        if [ x$feature_platform_search_hint = xy ]; then
          search --no-floppy --fs-uuid --set=root --hint='hd0,gpt2'  94ac5f77-cb8a-...
        else
          search --no-floppy --fs-uuid --set=root 94ac5f77-cb8a-495e-a65b-2ef7442b837c
        fi
        linux16 /vmlinuz-3.10.0-229.el7.x86_64 root=/dev/mapper/centos-root ro rd.lvm.lv
            =centos/root rd.lvm.lv=centos/swap crashkernel=auto rhgb quiet
            systemd.unit=rescue.target
        initrd16 /initramfs-3.10.0-229.el7.x86_64.img
}

[root@study ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
```

最后一步当然不要忘记重建你的 grub.cfg 啰！然后重新开机测试一下，如果一切顺利，你会发现如下的画面：



图19.3.4、默认的菜单环境

你直接在 1, 2, 3 菜单上面按下 enter 就可以顺利的继续开机，而不用输入任何的密码，这是因为有 --unrestricted 参数的关系。 第 4, 5 菜单中，如果你按下 enter 的话，就会出现如下画面：

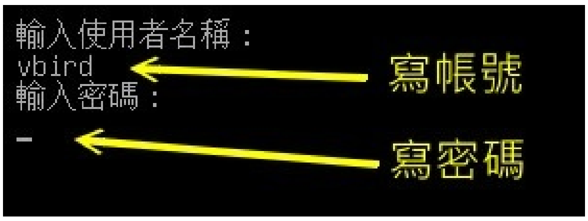


图19.3.5、需要输入帐号密码的环境

你可能会怀疑，怪了！为啥 4, 5 需要输入密码才行？而且一定要 vbird 这个系统管理员的密码才可接受？使用 dmstai 就不可以！ 这是因为我们在 4, 5 忘记加上 --users 也忘记加上 --restricted 了！因此这两个项目“一定要系统管理员”才能够进入与修改。

最后，你在第 6 个菜单上面输入 e 来想要修改参数时，输入的帐密确实是 dmtsai 的帐密，但是，就是无法修改参数耶！怎么回事啊？ 我们前面讲过了， grub2 两个基本的功能 （1）修改参数与 （2）进入菜单开机模式，只有系统管理员能够修改参数，一般用户只能选择可用的开机菜单啦！ 这样说，终于理解了吧？哈哈！

问：我的默认菜单里面没有加上 --unrestricted 项目，同时已经设置了 set superusers="vbird" 了， 那请教一下，开机的时候能不能顺利开机 （没有输入帐密的情况下？）答：因为没有写上 --unrestricted 的项目，同时又加上了 superusers="vbird" 的设置项目，这表示“ grub.cfg 内的所有参数都已经受到限制”了， 所以，当倒数读秒结束后，系统会叫出帐号密码输入的窗口给你填写，如果没有填写就会一直卡住了！因此无法顺利开机喔！

19.4 开机过程的问题解决

很多时候，我们可能因为做了某些设置，或者是因为不正常关机（例如未经通知的停电等等）而导致系统的 filesystem 错乱，此时，Linux 可能无法顺利开机成功，那怎么办呢？难道要重灌？当然不需要啦！进入 rescue 模式去处理处理，应该就 OK 的啦！下面我们就来谈一谈如何处理几个常见的问题！

19.4.1 忘记 root 密码的解决之道

大家都知道鸟哥的记忆力不佳，容易忘东忘西的，那如果连 root 的密码都忘记了，怎么办？其实在 Linux 环境中 root 密码忘记时还是可以救回来的！只要能够进入并且挂载 /， 然后重新设置一下 root 的密码，就救回来啦！

只是新版的 systemd 的管理机制中，默认的 rescue 模式是无法直接取得 root 权限的喔！还是得要使用 root 的密码才能够登陆 rescure 环境耶！ 天哪！那怎办？没关系，还是有办法滴～通过一个名为“ rd.break ”的核心参数来处理即可喔！只是需要注意的是， rd.break 是在 Ram Disk 里面的操作系统状态，因此你不能直接取得原本的 linux 系统操作环境。所以，还需要 chroot 的支持！ 更由于 SELinux 的问题，你可能还得要加上某些特殊的流程才能顺利的搞定 root 密码的救援喔！

现在就让我们来实作一下吧！（1）按下 systemctl reboot 来重新开机，（2）进入到开机画面，在可以开机的菜单上按下 e 来进入编辑模式， 然后就在 linux16 的那个核心项目上面使用这个参数来处理：



图19.4.1、通过 rd.break 尝试救援 root 密码

改完之后按下 [crtl]+x 开始开机，开机完成后屏幕会出现如下的类似画面，此时请注意，你应该是在 RAM Disk 的环境，并不是原本的环境， 因此根目录下面的东西跟你原本的系统无关喔！而且，你的系统应该会被挂载到 /sysroot 目录下，因此，你得要这样作：

```
Generating "/run/initramfs/rdsosreport.txt"

Enter emergency mode. Exit the shell to continue.
Type "journalctl" to view system logs.
You might want to save "/run/initramfs/rdsosreport.txt" to a USB stick or /boot
after mounting them and attach it to a bug report.

switch_root:/#         # 无须输入密码即可取得 root 权限！
switch_root:/# mount   # 检查一下挂载点！一定会发现 /sysroot 才是对的！
.....（前面省略）.....
/dev/mapper/centos-root on /sysroot type xfs （ro,relatime,attr,inode64,noquota）

switch_root:/# mount -o remount,rw /sysroot  # 要先让它挂载成可读写！
switch_root:/# chroot /sysroot               # 实际切换了根目录的所在！取回你的环境了！

sh-4.2# echo "your_root_new_pw" &#124; passwd --stdin root
sh-4.2# touch /.autorelabel                  # 很重要！变回 SELinux 的安全本文～
sh-4.2# exit

switch_root:/# reboot
```

上述的流程你应该没啥大问题才对～比较不懂的，应该是 （1）chroot 是啥？ （2）为何需要 /.autorelabel 这个文件？

- chroot 目录：代表将你的根目录“暂时”切换到 chroot 之后所接的目录。因此，以上表为例，那个 /sysroot 将会被暂时作为根目录， 而我们知道那个目录其实就是最原先的系统根目录，所以你当然就能够用来处理你的文件系统与相关的帐号管理啰！
- 为何需要 /.autorelabel：在 rd.break 的 RAM Disk 环境下，系统是没有 SELinux 的，而你刚刚更改了 /etc/shadow （因为改密码啊！）， 所以“这个文件的 SELinux 安全本文的特性将会被取消”喔！如果你没有让系统于开机时自动的回复 SELinux 的安全本文， 你的系统将产生“无法登陆”的问题 （在 SELinux 为 Enforcing 的模式下！）加上 /.autorelabel 就是要让系统在开机的时候自动的使用默认的 SELinux type 重新写入 SELinux 安全本文到每个文件去！。

不过加上 /.autorelabel 之后，系统在开机就会重新写入 SELinux 的 type 到每个文件，因此会花不少的时间喔！如果你不想要花太多时间， 还有个方法可以处理：

- 在 rd.break 模式下，修改完 root 密码后，将 /etc/selinux/config 内的 SELinux 类型改为 permissive
- 重新开机后，使用 root 的身份下达“ restorecon -Rv /etc ”仅修改 /etc 下面的文件；
- 重新修改 /etc/selinux/config 改回 enforcing ，然后“ setenforce 1 ”即可！

19.4.2 直接开机就以 root 执行 bash 的方法

除了上述的 rd.break 之外，我们还可以直接开机取得系统根目录后，让系统直接丢一个 bash 给我们使用喔！ 使用的方法很简单，就同样在开机的过程中，同在 linux16 的那一行，最后面不要使用 rd.break 而是使用“ init=/bin/bash ”即可！ 最后开机完成就会丢一个 bash 给我们！同样不需要 root 密码而有 root 权限！

但是要完整的操作该系统是不可能的，因为我们将 PID 一号更改为 bash 啦！所以，最多还是用在救援方面就是了！ 而且，同样的，要操作该系统你还是得要 remount 根目录才行啊！否则无法更改文件系统啦！基本上，这个系统的处理方法你应该是要这样作的：



```
                                        for 0xbffff000-0xc0000000, requested 0x10, got 0x0
bash-4.2# mount -o remount,rw /
bash-4.2# echo "your_root_pw" | passwd --stdin root
Changing password for user root.
passwd: all authentication tokens updated successfully.
bash-4.2# reboot
bash: reboot: command not found
bash-4.2# /sbin/reboot
Failed to talk to init daemon.
bash-4.2# pstree -p
bash(1)---pstree(472)
bash-4.2# _
```

图19.4.2、直接开机使用 bash 的方法

如上图的完整截图，你会发现由于是最默认的 bash 环境，所以连 PATH 都仅有 /bin 而已～所以你不能下达 reboot ！同时， 由于没有 systemd 或者是 init 的存在，所以真的使用绝对路径来下达 reboot 时，系统也是无法协助你重新开机啦！ 此时只能按下 reset 或者是强制关机后，才能再次开机！所以...感觉上还是 rd.break 比较保险...

同时请注意，鸟哥上面刻意忘记处理 /.autorelabel 的文件创建～你如果按照鸟哥上述的方法实作的话，嘿嘿！此时应该是无法登陆的喔！ 请重新开机进入 rd.break 模式，然后使用 SELinux 改为 permissive 的方法来实验看看。等到可以顺利以 root 登陆系统后， 使用 restorecon -Rv /etc 来瞧一瞧，应该会像下面这样：

```
[root@study ~]# getenforce
Permissive

[root@study ~]# restorecon -Rv /etc
restorecon reset /etc/shadow context system_u:object_r:unlabeled_t:s0
   -&gt;system_u:object_r:shadow_t:s0
restorecon reset /etc/selinux/config context system_u:object_r:unlabeled_t:s0
   -&gt;system_u:object_r:selinux_config_t:s0

[root@study ~]# vim /etc/selinux/config
SELINUX=enforcing

[root@study ~]# setenforce 1
```

19.4.3 因文件系统错误而无法开机

如果因为设置错误导致无法开机时，要怎么办啊？这就更简单了！最容易出错的设置而导致无法顺利开机的步骤，通常就是 /etc/fstab 这个文件了，尤其是使用者在实作 Quota/LVM/RAID 时，最容易写错参数， 又没有经过 mount -a 来测试挂载，就立刻直接重新开机，真要命！无法开机成功怎么办？这种情况的问题大多如下面的画面所示：



图 19.4.3、文件系统错误的示意图

看到最后两行，他说可以输入 root 的密码继续加以救援喔！那请输入 root 的密码来取得 bash 并以 mount -o remount,rw / 将根目录挂载成可读写后，继续处理吧！其实会造成上述画面可能的原因除了 /etc/fstab 编辑错误之外，如果你曾经不正常关机后，也可能导致文件系统不一致 （Inconsistent） 的情况， 也有可能会出现相同的问题啊！如果是扇区错乱的情况，请看到上图中的第二行处， fsck 告知其实是 /dev/md0 出错， 此时你就应该要利用 fsck.ext3 去检测 /dev/md0 才是！等到系统发现错误，并且出现“clear [Y/N]”时，输入“ y ”吧！

当然啦，如果是 XFS 文件系统的话，可能就得要使用 xfs_repair 这个指令来处理。这个 fsck/xfs_repair 的过程可能会很长，而且如果你的 partition 上面的 filesystem 有过多的数据损毁时，即使 fsck/xfs_repair 完成后，可能因为伤到系统盘，导致某些关键系统文件数据的损毁，那么依旧是无法进入 Linux 的。此时，就好就是将系统当中的重要数据复制出来，然后重新安装，并且检验一下，是否实体硬盘有损伤的现象才好！不过一般来说，不太可能会这样啦～ 通常都是文件系统处理完毕后，就能够顺利再次进入 Linux 了。

19.5 重点回顾

- Linux 不可随意关机，否则容易造成文件系统错乱或者是其他无法开机的问题；
- 开机流程主要是：BIOS、MBR、Loader、kernel+initramfs、systemd 等流程
- Loader 具有提供菜单、载入核心文件、转交控制权给其他 loader 等功能。
- boot loader 可以安装在 MBR 或者是每个分区的 boot sector 区域中
- initramfs 可以提供核心在开机过程中所需要的最重要的模块，通常与磁盘及文件系统有关的模块；
- systemd 的配置文件为主要来自 /etc/systemd/system/default.target 项目；
- 额外的设备与模块对应，可写入 /etc/modprobe.d/*.conf 中；
- 核心模块的管理可使用 lsmod, modinfo, rmmod, insmod, modprobe 等指令；
- modprobe 主要参考 /lib/modules/$（uanem -r）/modules.dep 的设置来载入与卸载核心模块；
- grub2 的配置文件与相关文件系统定义文件大多放置于 /boot/grub2 目录中，配置文件名为 grub.cfg
- grub2 对磁盘的代号设置与 Linux 不同，主要通过侦测的顺序来给予设置。如 （hd0） 及 （hd0,1） 等。
- grub.cfg 内每个菜单与 menuentry 有关，而直接指定核心开机时，至少需要 linux16 及 initrd16 两个项目
- grub.cfg 内设置 loader 控制权移交时，最重要者为 chainloader +1 这个项目。
- 若想要重建 initramfs ，可使用 dracut 或 mkinitrd 处理
- 重新安装 grub2 到 MBR 或 boot sector 时，可以利用 grub2-install 来处理。
- 若想要进入救援模式，可于开机菜单过程中，在 linux16 的项目后面加入“ rd.break ”或“ init=/bin/bash ”等方式来进入救援模式。
- 我们可以对 grub2 的个别菜单给予不同的密码。

19.6 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ）

- 情境仿眞题一：利用救援光盘来处理系统的错误导致无法开机的问题。
    - 目标：了解救援光盘的功能；
    - 前提：了解 grub 的原理，并且知道如何使用 chroot 功能；
    - 需求：打字可以再加快一点啊！ ^_^ 这个部分鸟哥就不捉图了，请大家自行处理啰～假设你的系统出问题而无法顺利开机，此时拿出原版光盘，然后重新以光盘来启动你的系统。 然后你应该要这样作的：
    - 利用光盘开机时，看到开机项目后，请选择“Troubleshooting”项目 --> “Rescue a CentOS system”项目，按下 Enter 就开始开机程序；
    - 然后就进入救援光盘模式的文件系统搜寻了！这个救援光盘会去找出目前你的主机里面与 CentOS 7.x 相关的操作系统， 并将该操作系统汇整成为一个 chroot 的环境等待你的处置！但是他会有三个模式可以选择，分别是“continue”继续成为可读写挂载； “Read-Only”将侦测到的操作系统变成只读挂载；“Skip”略过这次的救援动作。在这里我们选择“ Continue ”吧！
    - 如果你有安装多个 CentOS 7.x 的操作系统 （多重操作系统的实作），那就会出现菜单让你选择想要处理的根目录是哪个！ 选择完毕就请按 Enter 吧！
    - 然后系统会将侦测到的信息通知你！一般来说，可能会在屏幕上显示类似这样的讯息：“ chroot /mnt/sysimage”此时请按下 OK 吧！
    - 按下 OK 后，系统会丢给你一个 shell 使用，先用 df 看一下挂载情况是否正确？ 若不正确请手动挂载其他未被挂载的 partition 。等到一切搞定后，利用 chroot /mnt/sysimage 来转成你原本的操作系统环境吧！等到你将一切出问题的地方都搞定，请 reboot 系统，且取出光盘，用硬盘开机吧！

简答题部分：

- 因为 root 密码忘记，我使用 rd.break 的核心参数重新开机，并且修改完 root 密码，重新开机后可以顺利开机完毕， 但是我使用所有的帐号却都无法登陆系统！为何会如此？可能原因为何？最可能的原因是 /.autorelabel 没有创建，且你为 SELinux Enforcing 的模式之故。如果是这样， 那你必须要重新进入 rd.break ，然后重新创建 /.autorelabel 即可。若不想要于开机过程等太久， 可以将 /etc/selinux/config 内的 SELinux 类型设置为 permissive 再以 19.4.1 的方法去 restorecon 回复 /etc 下面的文件 SELinux 类型即可。
- 万一不幸，我的一些模块没有办法让 Linux 的核心捉到，但是偏偏这个核心明明就有支

持该模块， 我要让该模块在开机的时候就被载入，那么应该写入那个文件？应该写入 /etc/modprobe.d/*.conf* 这个文件，他是模块载入相关的地方呢！当然，也可以写入 */etc/sysconfig/modules/* 里面。

- 如何在 grub2 开机过程当中，指定以“ multi-user.target ”来开机？在开机进入 boot loader 之后，利用 grub shell 的功能，亦即输入“ e ”进入编辑模式， 然后在 linux16 后面增加：linux16 .... systemd.unit=multi-user.target 就能够进入纯文本模式啰！
- 如果你不小心先安装 Linux 再安装 Windows 导致 boot loader 无法找到 Linux 的开机菜单，该如何挽救？ 方法有很多，例如：（1）借助第三方软件，安装类似 spfdisk 的软件在 MBR 里面，因为他同时认识 Linux 与 Windows ，所以就可以用他来进入 Linux 啦！（2）或者使用类似 KNOPPIX 的 Live CD 以光盘开机进入 Linux 之后，再以 chroot 软件切换根目录 (/) ，然后重新安装 grub 等 boot loader ，同样也可以重新让两个操作系统存在啦！ 总之，只要你知道 MBR / Super block / boot loader 之间的相关性，怎么切换都可能啊！ ^_^

19.7 参考资料与延伸阅读

- [1]BIOS 的 POST 功能解释：http://en.wikipedia.org/wiki/Power-on_self-test
- [2]BIOS 的 INT 13 硬件中断解释：http://en.wikipedia.org/wiki/INT_13
- [3]关于 splash 的相关说明：http://ruslug.rutgers.edu/~mcgrof/grub-images/
- [4]一些 grub 出错时的解决之道： http://wiki.linuxquestions.org/wiki/GRUB_boot_menu http://forums.gentoo.org/viewtopic.php?t=122656&highlight=grub+error+collection
- info grub （尤其是 6.1 的段落，在讲解 /etc/default/grub 的设置项目）
- GNU 官方网站关于 grub 的说明文档： http://www.gnu.org/software/grub/manual/html_node/
- 纯文本屏幕分辨率的修改方法： http://phorum.study-area.org/viewtopic.php?t=14776

2003/02/10：第一次完成 2005/09/19：将旧的文章移动到 此处 。 2005/09/26：将 核心编译一文订为进阶篇，不一定要学啦！但是核心模块不可不题，所以，新增一小节！2005/09/28：终于给他完成去！好累～ 2005/10/09：加上参考文献数据，以及修改一些些 kernel 开机时， grub 的 vga 设置值的解说。 2005/11/09：加上了关于较大硬盘所产生的困扰！ 2006/08/21：MBR 应该只有 512 Bytes ，结果误植为 512 KBytes ，抱歉！2007/06/27：新增 initrd 的说明，请参考这里。 2009/04/09：将旧的基于 FC4 的文章移动到此处。 2009/04/10：取消了 LILO 的 boot loader 说明！毕竟这玩意儿已经退流行！所以不再强调！有需要请查询此处。 2009/04/30：修订完毕，加强 init=/bin/bash 的说明，以及 grub 的密码管理！ 2009/09/14：加入情境仿眞，并根据讨论区 linuxfans 兄的建议，修改了一些地方！详情请参考讨论区建议！ 2015/08/20：将旧的基于 CentOS 5.x 的 grub 1.x 版本移动到这里啰！

第二十章、基础系统设置与备份策略

最近更新日期：20//

新的 CentOS 7 有针对不同的服务提供了相当大量的命令行设置模式，因此过去那个 setup 似乎没有什么用了！取而代之的是许多加入了 bash-complete 提供了不少参数补全的设置工具！甚至包括网络设置也是通过这个机制哩！我们这个小章节主要就是在介绍如何通过这些基本的指令来设置系统就是了。另外，万一不幸你的 Linux 被骇客入侵了、或是你的 Linux 系统由于硬件关系（不论是天灾还是人祸）而挂掉了！这个时候，请问如何快速的回复你的系统呢？呵呵！当然啰，如果有备份数据的话，那么回复系统所花费的时间与成本将降低相当的多！平时最好就养成备份的习惯，以免突然间的系统损毁造成手足无措！此外，哪些文件最需要备份呢？又，备份是需要完整的备份还是仅备份重要数据即可？嗯！确实需要考虑看看呦！

20.1 系统基本设置

我们的 CentOS 7 系统其实有很多东西需要来设置的，包括之前稍微谈过的语系、日期、时间、网络设置等等。 CentOS 6.x 以前有个名为 setup 的软件将许多的设置做成类图形界面，连防火墙都可以这样搞定！不过这个功能在 CentOS 7 已经式微～ 这是因为 CentOS 7 已经将很多的软件指令作的还不赖，又加入了 bash-complete 的功能，指令下达确实还 OK 啦！如果不习惯指令，很多的图形界面也可以使用～因此，setup 的需求就减少很多了！ 下面我们会介绍基本的系统设置需求，其实也是将之前章节里面稍微谈过个数据做个汇整就是了！

20.1.1 网络设置 （手动设置与DHCP自动取得）

网络其实是又可爱又麻烦的玩意儿，如果你是网络管理员，那么你必须要了解区域网络内的 IP, gateway, netmask 等参数，如果还想要连上 Internet ，那么就得要理解 DNS 代表的意义为何。如果你的单位想要拥有自己的网域名称， 那么架设 DNS 服务器则是不可或缺的。总之，要设置网络服务器之前，你得要先理解网络基础就是了！ 没有人愿意自己的服务器老是被攻击或者是网络问题层出不穷吧！^_^

但鸟哥这里的网络介绍仅止于当你是一部单机的 Linux 用户端，而非服务器！所以你的各项网络参数只要找到网络管理员， 或者是找到你的 ISP （Internet Service Provider） ，向他询问网络参数的取得方式以及实际的网络参数即可。 通常网络参数的取得方式在台湾常见的有下面这几种：

- 手动设置固定 IP

常见于学术网络的服务器设置、公司行号内的特定座位等。这种方式你必须要取得下面的几个参数才能够让你的 Linux 上网的：

- IP
- 子网络遮罩（netmask）
- 通讯闸（gateway）
- DNS 主机的 IP （通常会有两个，若记不住的话，硬背 168.95.1.1 即可）

- 网络参数可自动取得 （dhcp 协定自动取得）

常见于 IP 分享器后端的主机，或者是利用电视线路的缆线上网 （cable modem） ，或者是学校宿舍的网络环境等。 这种网络参数取得方式就被称为 dhcp ，你啥事都不需要知道，只要知道设置上网方式为 dhcp 即可。

- 台湾的光纤到府与 ADSL 宽带拨接

不论你的 IP 是固定的还是每次拨接都不相同 （被称为浮动式 IP） ，只要是通过光纤到府或宽带调制解调器“拨接上网”的，就是使用这种方式。 拨接上网虽然还是使用网卡连接到调制解调器上，不过，系统最终会产生一个替代调制解调器的网络接口 （ppp0） ， 那个 ppp0 也

是一个实体网络接口啦！

不过，因为台湾目前所谓的“光世代”宽带上网的方式所提供的调制解调器中，内部已经涵盖了IP 分享与自动拨接功能， 因此，其实你在调制解调器后面也还是只需要“自动取得IP”的方式来取得网络参数即可喔！

了解了网络参数的取得方法后，你还得要知道一下我们通过啥硬件连上 Internet 的呢？其实就是网卡嘛。 目前的主流网卡为使用以太网络协定所开发出来的以太网卡 （Ethernet），因此我们 Linux 就称呼这种网络接口为 ethN （N 为数字）。 举例来说，鸟哥的这部测试机上面有一张以太网卡，因此鸟哥这部主机的网络接口就是 eth0 啰 （第一张为 0 号开始）。

不过新的 CentOS 7 开始对于网卡的编号则有另一套规则，网卡的界面代号现在与网卡的来源有关～基本上的网卡名称会是这样分类的：

- eno1 ：代表由主板 BIOS 内置的网卡
- ens1 ：代表由主板 BIOS 内置的 PCI-E 界面的网卡
- enp2s0 ：代表 PCI-E 界面的独立网卡，可能有多个插孔，因此会有 s0, s1... 的编号～
- eth0 ：如果上述的名称都不适用，就回到原本的默认网卡编号

其实不管什么网卡名称啦！想要知道你有多少网卡，直接下达“ ifconfig -a ”全部列出来即可！此外，CentOS 7 也希望我们不要手动修改配置文件， 直接使用所谓的 nmcli 这个指令来设置网络参数即可～因为鸟哥的测试机器是虚拟机，所以上述的网卡代号只有 eth0 能够支持～ 你得要自己看自己的系统上面的网卡代号才行喔！

- 手动设置 IP 网络参数

假设你已经向你的 ISP 取得你的网络参数，基本上的网络参数需要这些数据的：

- method: manual （手动设置）
- IP: 172.16.1.1
- netmask: 255.255.0.0
- gateway: 172.16.200.254
- DNS: 172.16.200.254
- hostname: study.centos.vbird

上面的数据除了 hostname 是可以暂时不理会的之外，如果你要上网，就得要有上面的这些数据才行啊！然后通过 nmcli 来处理！ 你得要先知道的是，nmcli 是通过一个名为“连线代号”的名称来设置是否要上网，而每个“连线代号”会有个“网卡代号”， 这两个东西通常设置成相同就是了。那就来先查查看目前系统上默认有什么连线代号吧！

```
[root@study ~]# nmcli connection show [网卡代号]
[root@study ~]# nmcli connection show
NAME  UUID                                  TYPE            DEVICE
eth0  505a7445-2aac-45c8-92df-dc10317cec22  802-3-ethernet  eth0
# NAME    就是连线代号，通常与后面的网卡 DEVICE 会一样！
# UUID    这个是特殊的设备识别，保留就好不要理他！
# TYPE    就是网卡的类型，通常就是以太网卡！
# DEVICE  当然就是网卡名称啰！
# 从上面我们会知道有个 eth0 的连线代号，那么来查察这个连线代号的设置为何？

[root@study ~]# nmcli connection show eth0
connection.id:                          eth0
connection.uuid:                        505a7445-2aac-45c8-92df-dc10317cec22
connection.interface-name:              eth0
connection.type:                        802-3-ethernet
connection.autoconnect:                 yes
.....（中间省略）.....
ipv4.method:                            manual
ipv4.dns:
ipv4.dns-search:
ipv4.addresses:                         192.168.1.100/24
ipv4.gateway:                           --
.....（中间省略）.....
IP4.ADDRESS[1]:                         192.168.1.100/24
IP4.GATEWAY:
IP6.ADDRESS[1]:                         fe80::5054:ff:fedf:e174/64
IP6.GATEWAY:
```

如上表的输出，最下面的大写的 IP4, IP6 指的是目前的实际使用的网络参数，最上面的 connection 开头的部份则指的是连线的状态！ 比较重要的参数鸟哥将它列出来如下：

- connection.autoconnect [yes|no] ：是否于开机时启动这个连线，默认通常是 yes 才对！
- ipv4.method [auto|manual] ：自动还是手动设置网络参数的意思
- ipv4.dns [dns_server_ip] ：就是填写 DNS 的 IP 位址～
- ipv4.addresses [IP/Netmask] ：就是 IP 与 netmask 的集合，中间用斜线 / 来隔开～
- ipv4.gateway [gw_ip] ：就是 gateway 的 IP 位址！

所以，根据上面的设置项目，我们来将网络参数设置好吧！

```
[root@study ~]# nmcli connection modify eth0 \
&gt;  connection.autoconnect yes \
&gt;  ipv4.method manual \
&gt;  ipv4.addresses 172.16.1.1/16 \
&gt;  ipv4.gateway 172.16.200.254 \
&gt;  ipv4.dns 172.16.200.254
# 上面只是“修改了配置文件”而已，要实际生效还得要启动 （up） 这个 eth0 连线界面才行喔！

[root@study ~]# nmcli connection up eth0
[root@study ~]# nmcli connection show eth0
.....（前面省略）.....
IP4.ADDRESS[1]:                         172.16.1.1/16
IP4.GATEWAY:                            172.16.200.254
IP4.DNS[1]:                             172.16.200.254
IP6.ADDRESS[1]:                         fe80::5054:ff:fedf:e174/64
IP6.GATEWAY:
```

最终执行“ nmcli connection show eth0 ”然后看最下方，是否为正确的设置值呢？如果是的话，那就万事 OK 啦！

- 自动取得 IP 参数

如果你的网络是由自动取得的 DHCP 协定所分配的，那就太棒了！上述的所有功能你通通不需要背～只需要知道 ipv4.method 那个项目填成 auto 即可！ 所以来查察，如果变成自动取得，网络设置要如何处理呢？

```
[root@study ~]# nmcli connection modify eth0 \
&gt;  connection.autoconnect yes \
&gt;  ipv4.method auto

[root@study ~]# nmcli connection up eth0
[root@study ~]# nmcli connection show eth0
IP4.ADDRESS[1]:                         172.16.2.76/16
IP4.ADDRESS[2]:                         172.16.1.1/16
IP4.GATEWAY:                            172.16.200.254
IP4.DNS[1]:                             172.16.200.254
```

自动取得 IP 要简单太多了！同时下达 modify 之后，整个配置文件就写入了！因此你无须使用 vim 去重新改写与设置！ 鸟哥是认为， nmcli 确实不错用喔！另外，上面的参数中，那个 connection..., ipv4... 等等的，你也可以使用 [tab] 去调用出来喔！ 也就是说， nmcli 有支持 bash-complete 的功能，所以指令下达也很方便的！

- 修改主机名称

主机名称的修改就得要通过 hostnamectl 这个指令来处理了！

```
[root@study ~]# hostnamectl [set-hostname 你的主机名]

# 1\. 显示目前的主机名称与相关信息
[root@study ~]# hostnamectl
   Static hostname: study.centos.vbird                # 这就是主机名称
         Icon name: computer
           Chassis: n/a
        Machine ID: 309eb890d09f440681f596543d95ec7a
           Boot ID: b2de392ff1f74e568829c716a7166ecd
    Virtualization: kvm
  Operating System: CentOS Linux 7 （Core）             # 操作系统名称！
       CPE OS Name: cpe:/o:centos:centos:7
            Kernel: Linux 3.10.0-229.el7.x86_64       # 核心版本也提供！
      Architecture: x86_64                            # 硬件等级也提供！

# 2\. 尝试修改主机名称为 www.centos.vbird 之后再改回来～
[root@study ~]# hostnamectl set-hostname www.centos.vbird
[root@study ~]# cat /etc/hostname
www.centos.vbird

[root@study ~]# hostnamectl set-hostname study.centos.vbird
```

20.1.2 日期与时间设置

在第四章的 date 指令解释中，我们曾经谈过这家伙可以进行日期、时间的设置。 不过，如果要改时区呢？例如台湾时区改成日本时区之类的，该如何处理？另外，真的设置了时间，那么下次开机可以是正确的时间吗？还是旧的时间？ 我们也知道有“网络校时”这个功能，那如果有网络的话，可以通过这家伙来校时吗？这就来谈谈。

- 时区的显示与设置

因为地球是圆的，每个时刻每个地区的时间可能都不一样。为了统一时间，所以有个所谓的“GMT、格林威治时间”这个时区！ 同时，在太平洋上面还有一条看不见的“换日线”哩！台湾地区就比格林威治时间多了 8 小时，因为我们会比较早看到太阳啦！ 那我怎么知道目前的时区设置是正确的呢？就通过 timedatectl 这个指令吧！

```
[root@study ~]# timedatectl [commamd]
选项与参数：
list-timezones ：列出系统上所有支持的时区名称
set-timezone   ：设置时区位置
set-time       ：设置时间
set-ntp        ：设置网络校时系统

# 1\. 显示目前的时区与时间等信息
[root@study ~]# timedatectl
      Local time: Tue 2015-09-01 19:50:09 CST  # 本地时间
  Universal time: Tue 2015-09-01 11:50:09 UTC  # UTC 时间，可称为格林威治标准时间
        RTC time: Tue 2015-09-01 11:50:12
        Timezone: Asia/Taipei （CST, +0800）     # 就是时区啰！
     NTP enabled: no
NTP synchronized: no
 RTC in local TZ: no
      DST active: n/a

# 2\. 显示出是否有 New_York 时区？若有，则请将目前的时区更新一下
[root@study ~]# timedatectl list-timezones &#124; grep -i new
America/New_York
America/North_Dakota/New_Salem

[root@study ~]# timedatectl set-timezone "America/New_York"
[root@study ~]# timedatectl
      Local time: Tue 2015-09-01 07:53:24 EDT
  Universal time: Tue 2015-09-01 11:53:24 UTC
        RTC time: Tue 2015-09-01 11:53:28
        Timezone: America/New_York （EDT, -0400）

[root@study ~]# timedatectl set-timezone "Asia/Taipei"
# 最后还是要记得改回来台湾时区喔！不要忘记了！
```

- 时间的调整

由于鸟哥的测试机使用的是虚拟机，默认虚拟机使用的是 UTC 时间而不是本地时间，所以在默认的情况下，测试机每次开机都会快上 8 小时... 所以就需要来调整一下时间啰！时间的格式可以是“ yyyy-mm-dd HH:MM ”的格式！比较方便记忆喔！

```
# 1\. 将时间调整到正确的时间点上！
[root@study ~]# timedatectl set-time "2015-09-01 12:02"
```

过去我们使用 date 去修改日期后，还得要使用 hwclock 去订正 BIOS 记录的时间～现在通过 timedatectl 一口气帮我们全部搞定，方便又轻松！

- 用 ntpdate 手动网络校时

其实鸟哥眞的不太爱让系统自动网络校时，比较喜欢自己手动网络校时。当然啦，写入 crontab 也是不错的想法～ 因为系统默认的自动校时会启动 NTP 协定相关的软件，会多开好几个 port ～想到就不喜欢的缘故啦！没啥特别的意思～ 那如何手动网络校时呢？很简单，通过 ntpdate 这个指令即可！

```
[root@study ~]# ntpdate tock.stdtime.gov.tw
 1 Sep 13:15:16 ntpdate[21171]: step time server 211.22.103.157 offset -0.794360 sec

[root@study ~]# hwclock -w
```

上述的 tock.stdtime.gov.tw 指的是台湾地区国家标准实验室提供的时间服务器，如果你在台湾本岛上，建议使用台湾提供的时间服务器来更新你的服务器时间， 速度会比较快些～至于 hwclock 则是将正确的时间写入你的 BIOS 时间记录内！如果确认可以执行，未来应该可以使用 crontab 来更新系统时间吧！

20.1.3 语系设置

我们在第四章知道有个 LANG 与 locale 的指令能够查询目前的语系数据与变量，也知道 /etc/locale.conf 其实就是语系的配置文件。 此外，你还得要知道的是，系统的语系与你目前软件的语系数据可能是可以不一样的！如果想要知道目前“系统语系”的话， 除了调用配置文件之外，也能够使用 localectl 来查阅：

```
[root@study ~]# localectl
   System Locale: LANG=zh_TW.utf8             # 下面这些数据就是“系统语系”
                  LC_NUMERIC=zh_TW.UTF-8
                  LC_TIME=zh_TW.UTF-8
                  LC_MONETARY=zh_TW.UTF-8
                  LC_PAPER=zh_TW.UTF-8
                  LC_MEASUREMENT=zh_TW.UTF-8
       VC Keymap: cn
      X11 Layout: cn
     X11 Options: grp:ctrl_shift_toggle

[root@study ~]# locale
LANG=zh_TW.utf8            # 下面的则是“当前这个软件的语系”数据！
LC_CTYPE="en_US.utf8"
LC_NUMERIC="en_US.utf8"
.....（中间省略）.....
LC_ALL=en_US.utf8
```

从上面的两个指令结果你会发现到，系统的语系其实是中文的万国码 （zh_TW.UTF8） 这个语系。不过鸟哥为了目前的教学文件制作， 需要取消中文的显示，而以较为单纯的英文语系来处理～因此使用 locale 指令时，就可以发现“鸟哥的 bash 使用的语系环境为 en_US.utf8”这一个！ 我们知道直接输入的 locale 查询到的语系，就是目前这个 bash 默认显示的语言，那你应该会觉得怪，那系统语系 （localectl） 显示的语系用在哪？

其实鸟哥一登陆系统时，取得的语系确实是 zh_TW.utf8 这一个的，只是通过“ export LC_ALL=en_US.utf8 ”来切换为英文语系而已。 此外，如果你有启用图形界面登陆的话，那么默认的显示语系也是通过这个 localectl 所输出的系统语系喔！

问：如果你跟着鸟哥的测试机器一路走来，图形界面将会是中文万国码的提示登陆字符。如何改成英文语系的登陆界面？答：就是将 locale 改成 en_US.utf8 之后，再转成图形界面即可！

```
[root@study ~]# localectl set-locale LANG=en_US.utf8
[root@study ~]# systemctl isolate multi-user.target
[root@study ~]# systemctl isolate graphical.target
```

接下来你就可以看到英文的登陆画面提示了！未来的默认语系也都会是英文界面喔！

20.1.4 防火墙简易设置

有网络没有防火墙还挺奇怪的，所以这个小节我们简单的来谈谈防火墙的一点点数据好了！

防火墙其实是一种网络数据的过滤方式，它可以依据你服务器启动的服务来设置是否放行，也能够针对你信任的用户来放行！ 这部份应该要对网络有点概念之后才来谈比较好，所以详细的数据会写入在服务器篇的内容。由于目前 CentOS 7 的默认防火墙机制为 firewalld， 他的管理界面主要是通过命令行 firewall-cmd 这个详细的指令～既然我们还没有谈到更多的防火墙与网络规则，想要了解 firewall-cmd 有点难！ 所以这个小节我们仅使用图形界面来介绍防火墙的相关数据而已！

要启动防火墙的图形管理界面，你当然就得要先登陆 X 才行！然后到“应用程序”-->“杂项”-->“防火墙”给它点下去，如下面的图示：

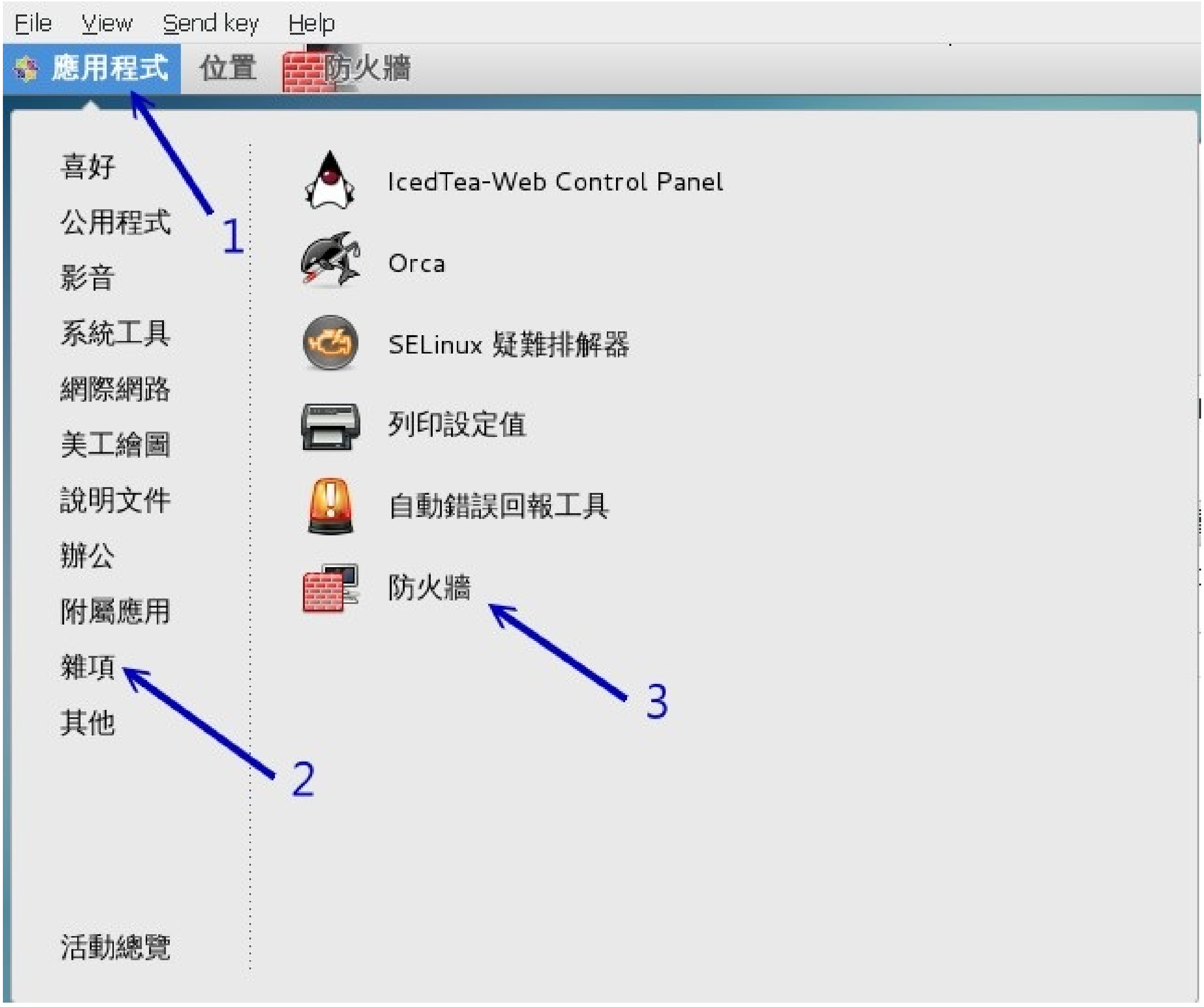


图20.1.1、防火墙启动的链接画面

之后出现的图形管理界面会有点像下面这样：

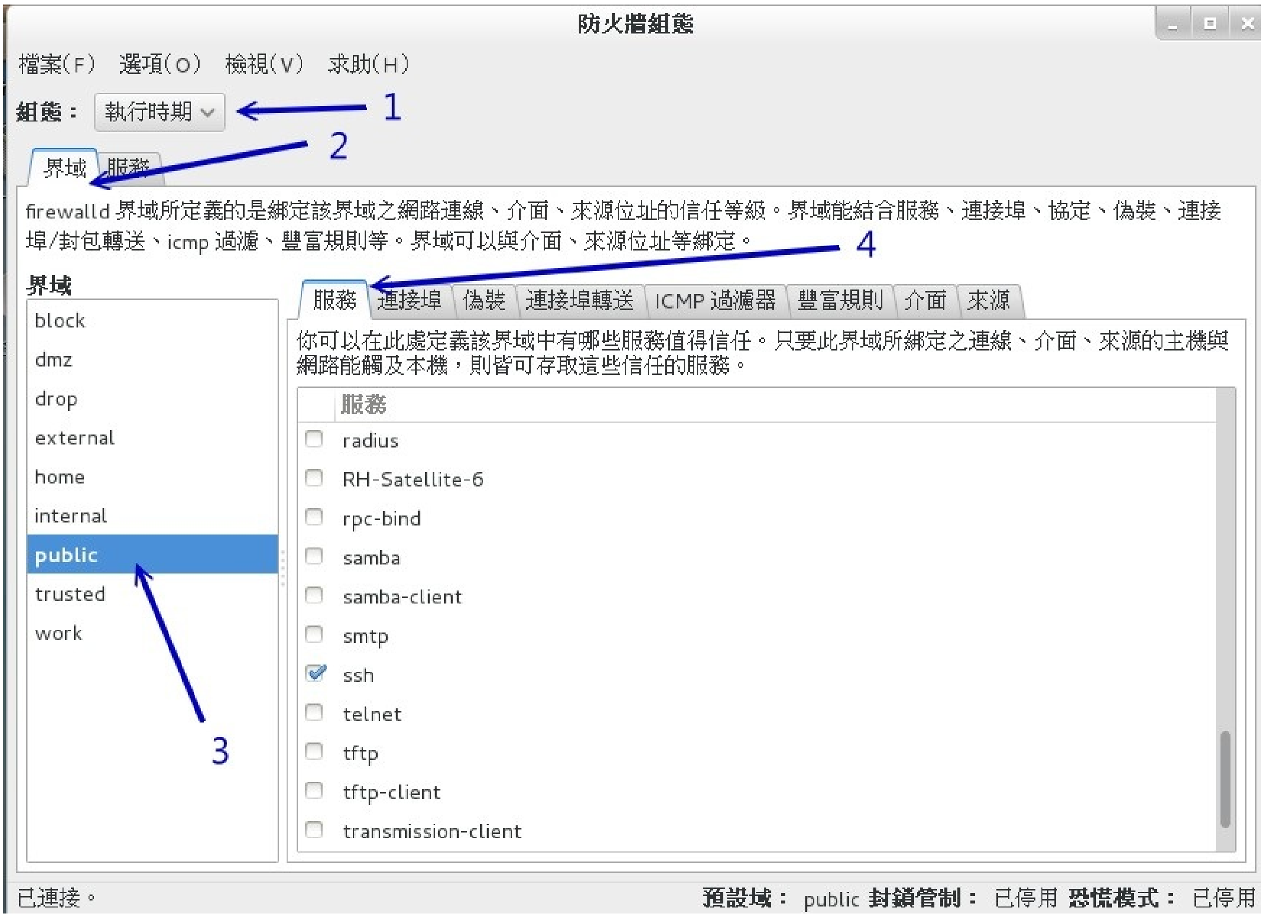


图20.1.2、防火墙图形管理界面示意图

- 组态：“执行时期”与“永久记录”的差异

如图 20.1.2 的箭头 1 处，基本上，防火墙的规则拟定大概有两种情况，一种是“暂时用来执行”的规则，一种则是“永久记录”的规则。 一般来说，刚刚启动防火墙时，这两种规则会一模一样。不过，后来可能你会暂时测试而加上几条规则，如果该规则没有写入“永久记录”区的话， 那下次重新载入防火墙时，该规则就会消失喔！所以请特别注意：“不要只是在执行阶段增加规则设置，而是必须要在永久记录区增加规则才行！”

- 界域 （zone） ：依据不同的环境所设计的网络界域 （zone）

玩过网络后，你可能会听过所谓的本机网络、NAT 与 DMZ 等网域，同时，可能还有可信任的 （trusted） 网域，或者是应该被抵挡 （drop/block） 的网域等等。 这些网域各有其功能～早期的 iptables 防火墙服务，所有的规则你都得要自己手动来撰写，然后规则的细分得要自己去规划， 所以很可能会导致一堆无法理解的规则。

新的 firewalld 服务就预先设计这些可能会被用到的网络环境，里面的规则除了 public （公开网域） 这个界域 （zone） 之外，其它的界域则暂时为没有启动的状况。 因此，在默认的情况下，如图 20.1.2 当中的 2 号箭头与 3 号箭头处，你只要考虑 public 那个项目即可！其他的领域等到读完服务器篇之后再来讨论。 所以，再说一次～你只要考虑 public 这个 zone 即可喔！

- 相关设置项目

接下来图 20.1.2 4 号箭头的地方就是重点啦！防火墙规则通常需要设置的地方有：

- 服务：一般来说，如果你的 Linux server 是作为 Internet 的服务器，提供的是比较一般的服务，那么只要处理“服务”项目即可。默认你的服务器已经提供了 ssh 与 dhcpv6-client 的服务端口喔！
- 端口：如果你提供的服务所启用的端口并不是正规的端口，举例来说，为了玩 systemd 与 SELinux 我们曾经将 ssh 的端口调整到 222 ，同时也曾经将 ftp 的端口调整到 555 对吧！那如果你想要让人家连进来，就不能只开放上面的“服务”项目，连这个“端口”的地方也需要调整才行！另外， 如果有某些比较特别的服务是 CentOS 默认没有提供的，所以“服务”当然也就没有存在！这时你也可以直接通过端口来搞定它！
- 丰富规则（rich rule）：如果你有“整个网域”需要放行或者是拒绝的时候，那么前两个项目就没有办法适用， 这时就得要这个项目来处理了。不过鸟哥测试了 7.1 这一版的设置，似乎怪怪的～因此，下面我们会以 firewall-cmd 来增加这一个项目的设置。
- 接口：就是这个界域主要是针对哪一个网卡来做规范的意思，我们只有一张网卡，所以当然就是 eth0 啰！

至于“伪装”、“端口转送”、“ICMP过滤器”、“来源”等等我们就不介绍了！毕竟那个是网络的东西，还不是在基础篇应该要告诉你的项目。 好了！现在假设我们的 Linux server 是要作为下面的几个重要的服务与相关的网域功能，你该如何设置防火墙呢？

- 要作为 ssh, www, ftp, https 等等正规端口的服务；
- 同时与前几章搭配，还需要放行 port 222 与 port 555 喔！
- 区域网络 192.168.1.0/24 这一段我们目前想要直接放行这段网域对我们服务器的连线

请注意，因为未来都要持续生效，所以请一定要去到“永久”的防火墙设置项目里头去处理！不然只有这次开机期间会生效而已～注意注意！ 好了，首先就来处理一下正规的服务端口的放行吧！不过因为永久的设置比较重要，因此你得要先经过授权认证才行！如下图所示。



图20.1.3、永久的设置需要权限的认证

注意如下图所示，你要先确认箭头 1, 2, 3 的地方是正确的，然后再直接勾选 ftp, http, https, ssh 即可！因为 ssh 默认已经被勾选， 所以鸟哥仅截图上头的项目而已！比较特别的是，勾选就生效～没有“确认”按钮喔！呵呵！相当有趣！

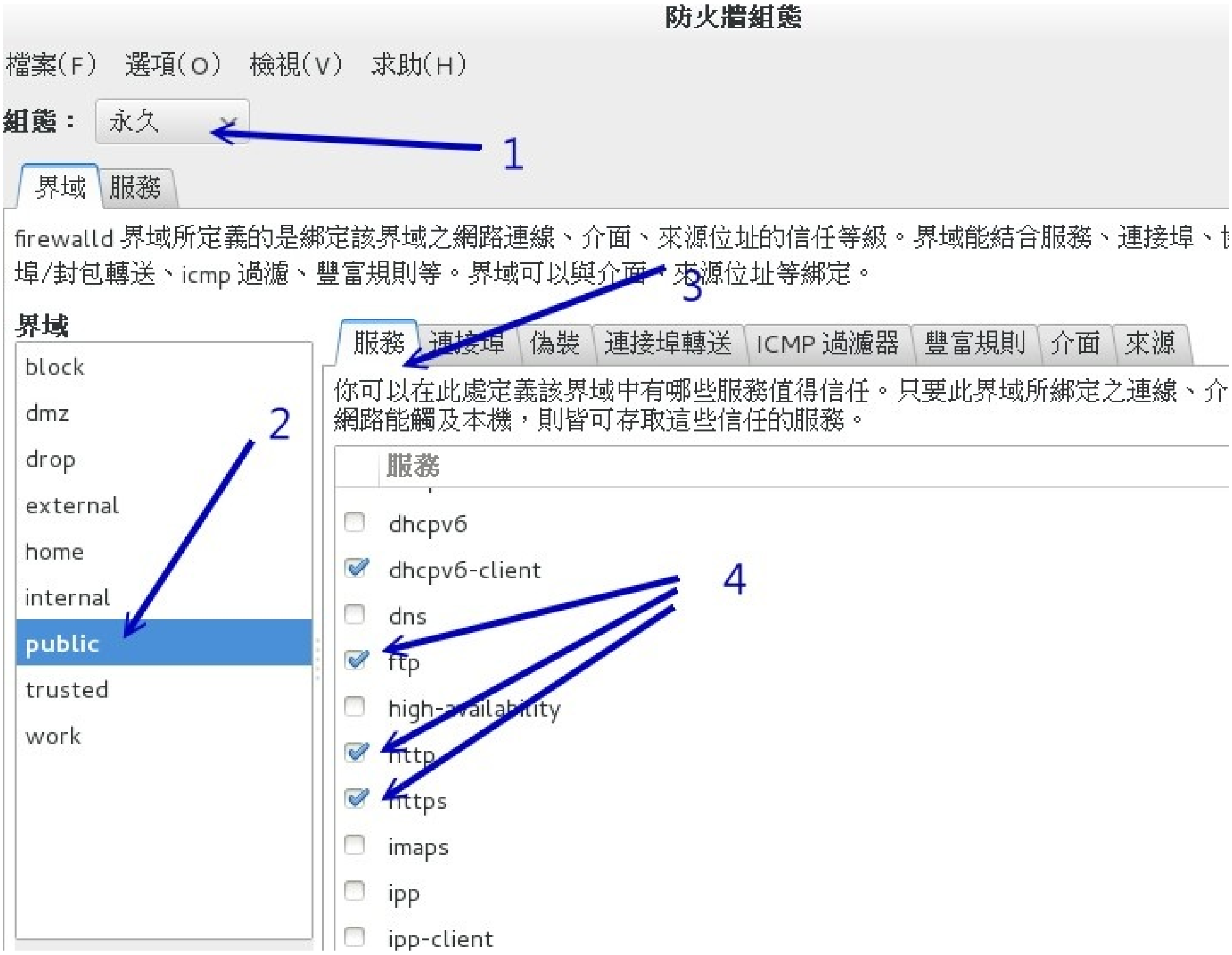


图20.1.4、以图形界面的方式放行正规服务的防火墙设置

接下来按下“端口”的页面，如下图所示，按下“加入”之后在出现的窗口当中填写你需要的端口号码，通常也就是 tcp 协定保留它不动！ 之后按下“确定”就好了！

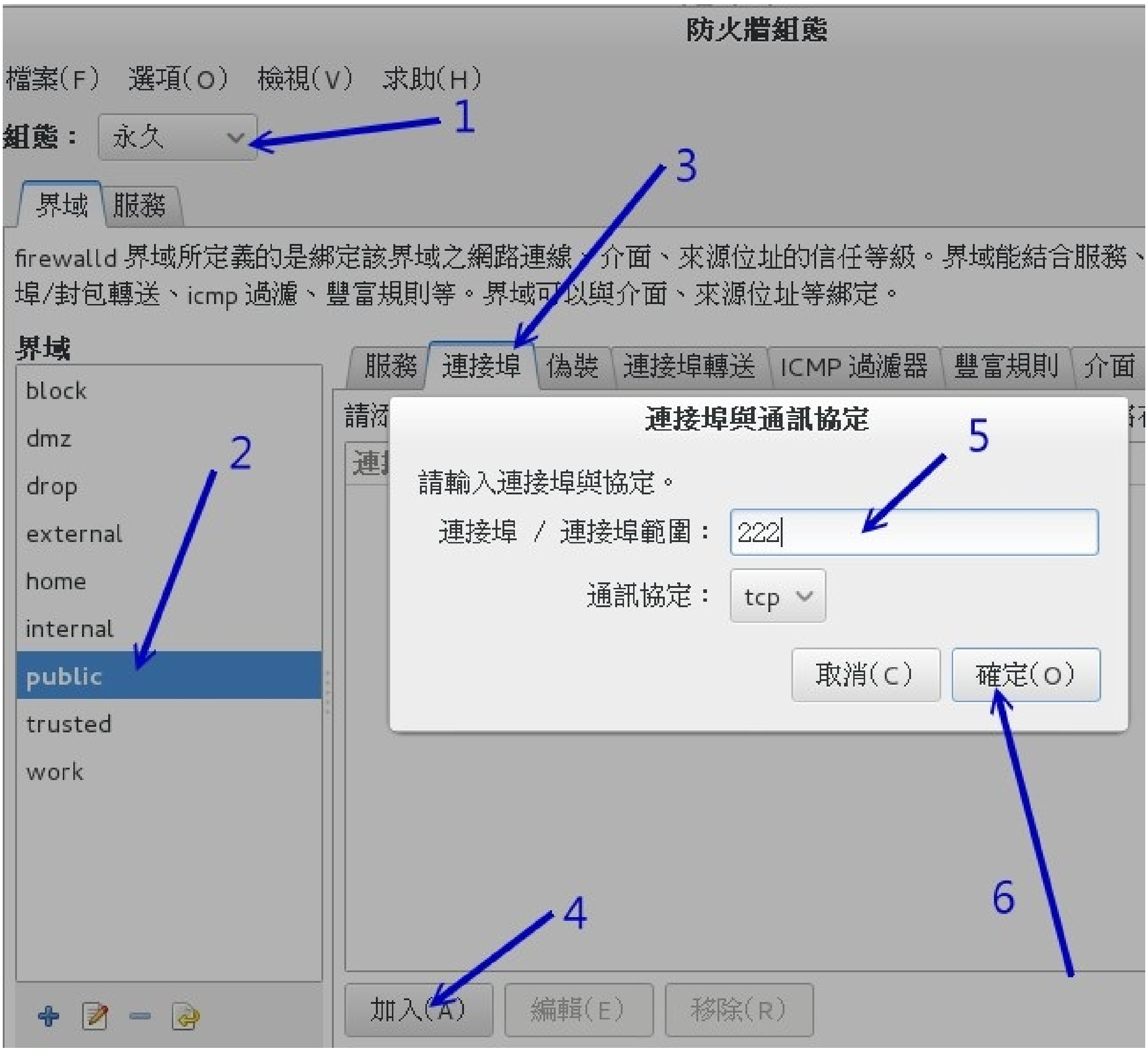


图20.1.5、以图形界面的方式放行部份非正规端口的防火墙设置

因为我们有两个端口要增加，所以请实作两次产生 222 与 555 的端口如下：



图20.1.6、以图形界面的方式放行部份非正规端口的防火墙设置

最后一个要处理的是区域网络的放行，我们刚刚谈到这个部份恐怕目前的图形界面软件有点怪异～所以，这时你可以这样下达指令即可！ 注意，下列的指令全部都是必要参数，只有 IP 网段的部份可以变动掉即可！

```
[root@study ~]# firewall-cmd --permanent --add-rich-rule='rule family="ipv4" \
&gt;  source address="192.168.1.0/24" accept'
success
[root@study ~]# firewall-cmd --reload
```

最后一行很重要喔！我们上面的图示通通是作用于“永久”设置中，只是变更配置文件，要让这些设置实际生效，那么就得要使用上面的 reload 项目， 让防火墙系统整个完整的再载入一下～那就 OK 啰！这样会使用简易的防火墙设置了吗？ ^_^

20.2 服务器硬件数据的收集

“工欲善其事，必先利其器”，这是一句大家耳熟能详的古人名言，在我们的信息设备上面也是一样的啊！ 在现在 （2015） 正好是 DDR3 切换到 DDR4 的时间点，假设你的服务器硬件刚刚好内存不太够，想要加内存， 那请教一下，你的主板插槽还够吗？你的内存需要 DDR3 还是 DDR4 呢？你的主机能不能吃到 8G 以上的单条内存？ 这就需要检查一下系统啰！不想拆机箱吧？那怎办？用软件去查啦！此外，磁盘会不会出问题？你怎么知道哪一颗磁盘出问题了？这就重要啦！

20.2.1 以系统内置 dmidecode 解析硬件配备

系统有个名为 dmidecode 的软件，这个软件挺有趣的，它可以解析 CPU 型号、主板型号与内存相关的型号等等～ 相当的有帮助！尤其是在升级配备上面！现在让我们来查一查鸟哥的虚拟机里头有啥东西吧！

```
[root@study ~]# dmidecode -t type
选项与参数：
详细的 type 项目请 man dmidecode 查询更多的数据，这里仅列出比较常用的项目：
1 ：详细的系统数据，含主板的型号与硬件的基础数据等
4 ：CPU 的相关数据，包括倍频、外频、核心数、核心绪数等
9 ：系统的相关插槽格式，包括 PCI, PCI-E 等等的插槽规格说明
17：每一个内存插槽的规格，若内有内存，则列出该内存的容量与型号

范例一：秀出整个系统的硬件信息，例如主板型号等等
[root@study ~]# dmidecode -t 1
# dmidecode 2.12
SMBIOS 2.4 present.

Handle 0x0100, DMI type 1, 27 Bytes
System Information
        Manufacturer: Red Hat
        Product Name: KVM
        Version: RHEL 6.6.0 PC
        Serial Number: Not Specified
        UUID: AA3CB5D1-4F42-45F7-8DBF-575445D3887F
        Wake-up Type: Power Switch
        SKU Number: Not Specified
        Family: Red Hat Enterprise Linux

范例二：那内存相关的数据呢？
[root@study ~]# dmidecode -t 17
# dmidecode 2.12
SMBIOS 2.4 present.

Handle 0x1100, DMI type 17, 21 Bytes
Memory Device
        Array Handle: 0x1000
        Error Information Handle: 0x0000
        Total Width: 64 bits
        Data Width: 64 bits
        Size: 3072 MB
        Form Factor: DIMM
        Set: None
        Locator: DIMM 0
        Bank Locator: Not Specified
        Type: RAM
        Type Detail: None
```

因为我们的系统是虚拟机，否则的话，你的主板型号、每一只安插的内存容量等等，都会被列出来在上述的画面中喔！ 这样可以让你了解系统的所有主要硬件配备为何！



Tips 因为某些缘故，鸟哥获得了一部机架式的服务器，不过该服务器就是内存不够。又因为某些缘故有朋友要送 ECC 的低电压内存给鸟哥！太开心了！ 不过为了担心内存与主板不相容，所以就使用了 dmidecode 去查主板型号，再到原厂网站查询相关主板规格，这才确认可以使用！感谢各位亲爱的朋友啊！！

20.2.2 硬件资源的收集与分析

现在我们知道系统硬件是由操作系统核心所管理的，由第十九章的开机流程分析中，我们也知道 Linux kernel 在开机时就能够侦测主机硬件并载入适当的模块来驱动硬件了。 而核心所侦测到的各项硬件设备，后来就会被记录在 /proc 与 /sys 当中了。 包括 /proc/cpuinfo, /proc/partitions, /proc/interrupts 等等。 更多的 /proc 内容介绍，先回到第十六章的程序管理瞧一瞧先！



Tips 其实核心所侦测到的硬件可能并非完全正确喔！因为他仅是“使用最适当的模块来驱动这个硬件”而已， 所以有时候难免会误判啦 （虽然概率非常之低）！那你可能想要以最新最正确的模块来驱动你的硬件， 此时，重新编译核心是一条可以达成的道路。不过，现在的 Linux 系统并没有很建议你一定要重新编译核心就是了。

那除了直接调用出 /proc 下面的文件内容之外，其实 Linux 有提供几个简单的指令来将核心所侦测到的硬件叫出来的～ 常见的指令有下面这些：

- gdisk：第七章曾经谈过，可以使用 gdisk -l 将分区表列出；
- dmesg：第十六章谈过， 观察核心运行过程当中所显示的各项讯息记录；
- vmstat：第十六章谈过，可分析系统 （CPU/RAM/IO） 目前的状态；
- lspci：列出整个 PC 系统的 PCI 接口设备！很有用的指令；
- lsusb：列出目前系统上面各个 USB 端口的状态，与连接的 USB 设备；
- iostat：与 vmstat 类似，可实时列出整个 CPU 与周边设备的 Input/Output 状态。

lspci, lsusb, iostat 是本章新谈到的指令，尤其如果你想要知道主板与各周边相关设备时，那个 lspci 眞是不可多得的好工具！而如果你想要知道目前 USB 插槽的使用情况以及侦测到的 USB 设备， 那个 lsusb 则好用到爆！至于 iostat 则是一个实时分析软件，与 vmstat 有异曲同工之妙！

基本上，想要知道你 Linux 主机的硬件配备，最好的方法还是直接拆开机箱去察看上面的信息 （这也是为何第零章会谈计概啊）！ 如果环境因素导致您无法直接拆开主机的话，那么直接 lspci 是很棒的一的方法：

- lspci

```
[root@study ~]# lspci [-vvn]
选项与参数：
-v   ：显示更多的 PCI 接口设备的详细信息；
-vv  ：比 -v 还要更详细的细部信息；
-n   ：直接观察 PCI 的 ID 而不是厂商名称

范例一：查阅您系统内的 PCI 总线相关设备：
[root@study ~]# lspci
00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)
00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]
00:01.1 IDE interface: Intel Corporation 82371SB PIIX3 IDE [Natoma/Triton II]
00:01.2 USB controller: Intel Corporation 82371SB PIIX3 USB [Natoma/Triton II] (rev 01)
00:01.3 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 03)
00:02.0 VGA compatible controller: Red Hat, Inc. QXL paravirtual graphic card (rev 04)
00:03.0 Ethernet controller: Red Hat, Inc Virtio network device
00:04.0 SCSI storage controller: Red Hat, Inc Virtio block device
00:05.0 RAM memory: Red Hat, Inc Virtio memory balloon
00:06.0 Audio device: Intel Corporation 82801FB/FBM/FR/FW/FRW (ICH6 Family) High Definit
        Controller (rev 01)
00:1d.0 USB controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #1 (
00:1d.1 USB controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #2 (
00:1d.2 USB controller: Intel Corporation 82801I (ICH9 Family) USB UHCI Controller #3 (
00:1d.7 USB controller: Intel Corporation 82801I (ICH9 Family) USB2 EHCI Controller #1
# 不必加任何的参数，就能够显示出目前主机上面的各个 PCI 接口的设备呢！
```

不必加上任何选项，就能够显示出目前的硬件配备为何。上面就是鸟哥的测试机所使用的主机配备。 包括使用 Intel 芯片的仿眞主板、南桥使用 ICH9 的控制芯片、附挂 QXL 的显卡、使用虚拟化的 Virtio 网卡等等。 您瞧瞧！很清楚，不是嘛。

如果你还想要了解某个设备的详细信息时，可以加上 -v 或 -vv 来显示更多的信息喔！ 举例来说，鸟哥想要知道那个以太网卡更详细的信息时，可以使用如下的选项来处理：

```
[root@study ~]# lspci -s 00:03.0 -vv
```

-s 后面接的那个怪东西每个设备的总线、插槽与相关函数功能啦！那个是我们硬件侦测所得到的数据啰！ 你可以对照下面这个文件来了解该串数据的意义：

- /usr/share/hwdata/pci.ids

其实那个就是 PCI 的标准 ID 与厂牌名称的对应表啦！此外，刚刚我们使用 lspci 时，其实所有的数据都是由 /proc/bus/pci/ 目录下的数据所取出的呢！了解了吧！^_^！不过，由于硬件的发展太过迅速，所以你的 pci.ids 文件可能会落伍了～那怎办？ 没关系～可以使用下面的方式来线上更新你的对应档：

```
[root@study ~]# update-pciids
```

- lsusb

刚刚谈到的是 PCI 接口设备，如果是想要知道系统接了多少个 USB 设备呢？那就使用 lsusb 吧！这个指令也是很简单的！

```
[root@study ~]# lsusb [-t]
选项与参数：
-t  ：使用类似树状目录来显示各个 USB 端口的相关性

范例一：列出目前鸟哥的测试用主机 USB 各端口状态
[root@study ~]# lsusb
Bus 002 Device 002: ID 0627:0001 Adomax Technology Co., Ltd
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub
# 如上所示，鸟哥的主机在 Bus 002 有接了一个设备，
# 该设备的 ID 是 0627:0001，对应的厂商与产品为 Adomax 的设备。
```

确实非常清楚吧！其中比较有趣的就属那个 ID 号码与厂商型号对照了！那也是写入在 /usr/share/hwdata/pci.ids 的东西，你也可以自行去查询一下喔！

- iostat

刚刚那个 lspci 找到的是目前主机上面的硬件配备，那么整部机器的储存设备， 主要是磁盘对吧！请问，您磁盘由开机到现在，已经存取多少数据呢？这个时候就得要 iostat 这个指令的帮忙了！



Tips 默认 CentOS 并没有安装这个软件，因此你必须要先安装他才行！如果你已经有网络了， 那么使用“ yum install sysstat ”先来安装此软件吧！否则无法进行如下的测试喔！

```
[root@study ~]# iostat [-c&#124;-d] [-k&#124;-m] [-t] [间隔秒数] [侦测次数]
选项与参数：
-c   ：仅显示 CPU 的状态；
-d   ：仅显示储存设备的状态，不可与 -c 一起用；
-k   ：默认显示的是 block ，这里可以改成 K Bytes 的大小来显示；
-m   ：与 -k 类似，只是以 MB 的单位来显示结果。
-t   ：显示日期出来；

范例一：显示一下目前整个系统的 CPU 与储存设备的状态
[root@study ~]# iostat
Linux 3.10.0-229.el7.x86_64 (study.centos.vbird)   09/02/2015   _x86_64_    (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.08    0.01    0.02    0.00    0.01   99.88

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda               0.46         5.42         3.16     973670     568007
scd0              0.00         0.00         0.00        154          0
sda               0.01         0.03         0.00       4826          0
dm-0              0.23         4.59         3.09     825092     555621
# 瞧！上面数据总共分为上下两部分，上半部显示的是 CPU 的当下信息；
# 下面数据则是显示储存设备包括 /dev/vda 的相关数据，他的数据意义：
# tps        ：平均每秒钟的传送次数！与数据传输“次数”有关，非容量！
# kB_read/s  ：开机到现在平均的读取单位；
# kB_wrtn/s  ：开机到现在平均的写入单位；
# kB_read    ：开机到现在，总共读出来的文件单位；
# kB_wrtn    ：开机到现在，总共写入的文件单位；

范例二：仅针对 vda ，每两秒钟侦测一次，并且共侦测三次储存设备
[root@study ~]# iostat -d 2 3 vda
Linux 3.10.0-229.el7.x86_64 (study.centos.vbird)   09/02/2015   _x86_64_    (4 CPU)

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda               0.46         5.41         3.16     973682     568148

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda               1.00         0.00         0.50          0          1

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
vda               0.00         0.00         0.00          0          0
# 仔细看一下，如果是有侦测次数的情况，那么第一次显示的是“从开机到现在的数据”，
# 第二次以后所显示的数据则代表两次侦测之间的系统传输值！举例来说，上面的信息中，
# 第二次显示的数据，则是两秒钟内（本案例）系统的总传输量与平均值。
```

通过 lspci 及 iostat 可以约略的了解到目前系统的状态还有目前的主机硬件数据呢！

20.2.3 了解磁盘的健康状态

其实 Linux server 最重要的就是“数据安全”了！而数据都是放在磁盘当中的，所以啰，无时无刻了解一下你的磁盘健康状况，应该是个好习惯吧！ 问题是，你怎么知道你的磁盘是好是坏啊？这时就得要来谈一个 smartd 的服务了！

SMART 其实是“ Self-Monitoring, Analysis and Reporting Technology System ”的缩写，主要用来监测目前常见的 ATA 与 SCSI 界面的磁盘， 只是，要被监测的磁盘也必须要支持 SMART 的协定才行！否则 smartd 就无法去下达指令，让磁盘进行自我健康检查～ 比较可惜的是，我们虚拟机的磁盘格式并不支持 smartd，所以无法用来作为测试！不过刚刚好鸟哥还有另外一颗用作 IDE 界面的 2G 磁盘， 这个就能够用来作为测试了！ (/dev/sda) ！

smartd 提供一只指令名为 smartctl，这个指令功能非常多！不过我们下面只想要介绍数个基本的操作，让各位了解一下如何确认你的磁盘是好是坏！

```
# 1\. 用 smartctl 来显示完整的 /dev/sda 的信息
[root@study ~]# smartctl -a /dev/sda
smartctl 6.2 2013-07-26 r3841 [x86_64-linux-3.10.0-229.el7.x86_64] （local build）
Copyright （C） 2002-13, Bruce Allen, Christian Franke, www.smartmontools.org

# 首先来输出一下这部磁盘的整体信息状况！包括制造商、序号、格式、SMART 支持度等等！
=== START OF INFORMATION SECTION ===
Device Model:     QEMU HARDDISK
Serial Number:    QM00002
Firmware Version: 0.12.1
User Capacity:    2,148,073,472 Bytes [2.14 GB]
Sector Size:      512 Bytes logical/physical
Device is:        Not in smartctl database [for details use: -P showall]
ATA Version is:   ATA/ATAPI-7, ATA/ATAPI-5 published, ANSI NCITS 340-2000
Local Time is:    Wed Sep  2 18:10:38 2015 CST
SMART support is: Available - device has SMART capability.
SMART support is: Enabled

=== START OF READ SMART DATA SECTION ===
SMART overall-health self-assessment test result: PASSED

# 接下来则是一堆基础说明！鸟哥这里先略过这段数据喔！
General SMART Values:
Offline data collection status:  （0x82） Offline data collection activity
                                        was completed without error.
                                        Auto Offline Data Collection: Enabled.
.....（中间省略）.....
# 再来则是有没有曾经发生过磁盘错乱的问题登录！
SMART Error Log Version: 1
No Errors Logged

# 当你下达过磁盘自我检测的过程，就会被记录在这里了！
SMART Self-test log structure revision number 1
Num  Test_Description    Status                  Remaining  LifeTime（hours）  LBA_of_firs
# 1  Short offline       Completed without error       00%      4660         -
# 2  Short offline       Completed without error       00%      4660         -

# 2\. 命令磁盘进行一次自我检测的动作，然后再次观察磁盘状态！
[root@study ~]# smartctl -t short /dev/sda
[root@study ~]# smartctl -a /dev/sda
.....（前面省略）.....
# 下面会多出一个第三笔的测试信息！看一下 Status 的状态，没有问题就是好消息！
SMART Self-test log structure revision number 1
Num  Test_Description    Status                  Remaining  LifeTime（hours）  LBA_of_firs
# 1  Short offline       Completed without error       00%      4660         -
# 2  Short offline       Completed without error       00%      4660         -
# 3  Short offline       Completed without error       00%      4660         -
```

不过要特别强调的是，因为进行磁盘自我检查时，可能磁盘的 I/O 状态会比较频繁，因此不建议在系统忙碌的时候进行喔！ 否则系统的性能是可能会被影响的哩！要注意！要注意！

20.3 备份要点

备份是个很重要的工作，很多人总是在系统损毁的时候才在哀嚎说：“我的数据啊！天那...！”此时才会发现备份数据的可爱！ 但是备份其实也非常可怕！因为你的重要数据都在备份文件里面，如果这个备份被窃取或遗失，其实对你的系统资安影响也非常大！ 同时，备份使用的媒体选择也非常多样，但是各种储存媒体各有其功能与优劣，所以当然得要选择啰！闲话少说，来谈谈备份吧！

20.3.1 备份数据的考虑

老实说，备份是系统损毁时等待救援的救星！因为你需要重新安装系统时， 备份的好坏会影响到你系统复原的进度！不过，我们想先知道的是，系统为什么会损毁啊？是人为的还是怎样产生的啊？ 事实上，系统有可能由于不预期的伤害而导致系统发生错误！ 什么是不预期的伤害呢？这是由于系统可能因为不预期的硬件损坏，例如硬盘坏掉等等，或者是软件问题导致系统出错， 包括人为的操作不当或是其他不明因素等等所致。下面我们就来谈谈系统损坏的情况与为何需要备份吧！

- 造成系统损毁的问题-硬件问题

基本上，“计算机是一个相当不可靠的机器”这句话在大部分的时间内还是成立的！ 常常会听到说“要计算机正常的工作，最重要的是要去拜拜！”嘿嘿！不要笑！这还是真的哩！ 尤其是在目前一些计算机周边硬件的生产良率 （就是将硬件产生出来之后，经过测试， 发现可正常工作的与不能正常工作的硬件总数之比值） 越来越差的情况之下，计算机的不稳定状态实在是越来越严重了！

一般来说，会造成系统损毁的硬件元件应该要算硬盘吧！因为其他的元件坏掉时，虽然会影响到系统的运行， 不过至少我们的数据还是存在硬盘当中的啊！为了避免这个困扰，于是乎有可备份用的 RAID1, RAID5, RAID6 等磁盘阵列的应用啊！但是如果是 RAID 控制芯片坏掉呢？这就麻烦了～所以说，如果有 RAID 系统时， 鸟哥个人还是觉得需要进行额外的备份才好的！如果数据够重要的话。

- 造成系统损毁的问题-软件与人的问题

根据分析，其实系统的软件伤害最严重的就属使用者的操作不当啦！像以前 Google 还没有这么厉害时， 人们都到讨论区去问问题，某些高手高手高高手被小白烦的不胜其扰，总是会回答：“喔！你的系统有问题喔！那请 rm -rf / 看看出现什么状况！ 做完再回来！”...你真的做下去就死定了！如果你的系统有这种小白管理员呢？敢不备份喔？

软件伤害除了来自主机上的使用者操作不当之外，最常见的可能是资安攻击事件了。 假如你的 Linux 系统上面某些 Internet 的服务软件是最新的！这也意味着可能是“相对最安全的”， 但是，这个世界目前的闲人是相当多的，你不知道什么时候会有所谓的“骇客软件”被提供出来，

万一你在 Internet 上面的服务程序被攻击，导致你的 Linux 系统全毁，这个时候怎么办？当然是要复原系统吧？

那如何复原被伤害的系统呢？“重新安装就好啦！”或许你会这么说， 但是，像鸟哥管理的几个网站的数据，尤其是 MySQL 数据库的数据，这些都是弥足珍贵的经验数据， 万一被损毁而救不回来的时候，不是很可惜吗？这个还好哩，万一你是某家银行的话， 那么数据的损毁可就不是能够等闲视之的！关系的可是数千甚至上万人的身家财产！这就是备份的重要性了！ 他可以最起码的稍微保障我们的数据有另外一份 copy 的备援以达到“安全回复”的基本要求！

- 主机角色不同，备份任务也不同

由于软硬件的问题都可能造成系统的损毁，所以备份当然就很重要啦！问题是，每一部主机都需要备份吗？ 多久备份一次呢？要备份什么数据呢？

早期有 ghost 这套单机备份软件，近期以来有台湾国家高速网络中心发展的再生龙（clonzilla） 软件， 这些软件的共同特性就是可以将你系统上面的磁盘数据完整的复制起来，变成一个大文件，你可以通过现在便宜到爆炸的 USB 外接磁盘来备份出来， 未来复原时，只要将 USB 安插到系统里面，就几乎可以进行裸机复原了哩！

但是，万一你的主机有提供 Internet 方面的服务呢？又该如何备份啊？举个例子来说，像是我们 Study Area 团队的讨论区网站 http://phorum.study-area.org 提供的是类似 BBS 的讨论文章， 虽然数据量不大，但是由于讨论区的文件是天天在增加的，每天都有相当多的信息流入， 由于某些信息都是属于重要的人物之留言，这个时候，我们能够让机器死掉吗？或者是能够一季三个月才备份一次吗？ 这个备份频率需求的考虑是非常重要的！

再提到 2002 年左右鸟哥的讨论区曾经挂点的问题，以及 2003 年初 Study-Area 讨论区挂点的问题，讨论区一旦挂点的话，该数据库内容如果损毁到无法救回来， 嘿嘿！要晓得讨论区可不是一个人的心血耶！有的时候 （像 Study-Area 讨论区） 是一群热心 Linux 的朋友们互相创建交流起来的数据流通网，如果死掉了，那么不是让这些热血青年的热情付之一炬了吗？ 所以啰，创建备份的策略 （频率、媒体、方法等） 是相当的重要的。

- 备份因素考虑

由于计算机 （尤其是目前的计算机，操作频率太高、硬件良率太差、使用者操作习惯不良、“某些”操作系统的当概率太高....） 的稳定性较差，所以啰！备份的工作就越来越重要了！ 那么一般我们在备份时考虑的因素有哪些呢？

- 备份哪些文件： 哪些数据对系统或使用者来说是重要的？那些数据就是值得备份的数据！例如 /etc/ 及 */home/* 等。
- 选择什么备份的媒介： 是可读写光盘、另一颗硬盘、同一颗硬盘的不同 partition、还是使用网络备援系统？ 哪一种的速度最快，最便宜，可将数据保存最久？这都可以考虑的。

- 考虑备份的方式： 是以完整备份（类似 ghost）来备份所有数据，还是使用差异备份仅备份有被更动过的数据即可？
- 备份的频率： 例如 Mariadb 数据库是否天天备份、若完整备份，需要多久进行一次？
- 备份使用的工具为何： 是利用 tar 、 cpio 、 dd 还是 dump 等等的备份工具？

下面我们就来谈一谈这些问题的解决之道吧！ ^_^

20.3.2 哪些 Linux 数据具有备份的意义

一般来说，鸟哥比较喜欢备份最重要的文件而已 （关键数据备份），而不是整个系统都备份起来 （完整备份, Full backup）！那么哪些文件是有必要备份的呢？具有备份意义的文件通常可以粗分为两大类，一类是系统基本设置信息、一类则是类似网络服务的内容数据。 那么各有哪些文件需要备份的呢？我们就来稍微分析一下。

- 操作系统本身需要备份的文件：

这方面的文件主要跟“帐号与系统配置文件”有关系！ 主要有哪些帐号的文件需要备份呢？就是 /etc/passwd, /etc/shadow, /etc/group, /etc/gshadow, /home 下面的使用者主文件夹等等，而由于 Linux 默认的重要参数文件都在 /etc/ 下面，所以只要将这个目录备份下来的话， 那么几乎所有的配置文件都可以被保存的！

至于 /home 目录是一般用户的主文件夹，自然也需要来备份一番！再来，由于使用者会有邮件吧！所以呢，这个 /var/spool/mail/ 内容也需要备份呦！另外，由于如果你曾经自行更动过核心，那么 /boot 里头的信息也就很重要啰！所以啰，这方面的数据你必须要备份的文件为：

- /etc/ 整个目录
- /home/ 整个目录
- /var/spool/mail/
- /var/spoll/{at|cron}/
- /boot/
- /root/
- 如果你自行安装过其他的软件，那么 /usr/local/ 或 /opt 也最好备份一下！

- 网络服务的数据库方面：

这部份的数据可就多而且复杂了，首先是这些网络服务软件的配置文件部分， 如果你的网络软件安装都是以原厂提供的为主，那么你的设置文件大多是在 /etc 下面，所以这个就没啥大问题！但若你的套件大多来自于自行的安装，那么 /usr/local 这个目录可就相当的重要了！

再来，每种服务提供的数据都不相同，这些数据很多都是人们提供的！举例来说，你的 WWW 服务器总是需要有人提供网页文件吧？否则浏览器来是要看啥咚咚？你的讨论区总是得要写入数据库系统吧？ 否则讨论的数据如何更新与记载？所以，使用者主动提供的文件，

以及服务运行过程会产生的数据， 都需要被考虑来备份。若我们假设我们提供的服务软件都是使用原厂的 RPM 安装的！所以要备份的数据文件有：

- 软件本身的设置文件，例如：/etc/ 整个目录，/usr/local/ 整个目录
- 软件服务提供的数据，以 WWW 及 Mariadb 为例： WWW 数据：/var/www 整个目录或 /srv/www 整个目录，及系统的使用者主文件夹 Mariadb ： /var/lib/mysql 整个目录
- 其他在 Linux 主机上面提供的服务之数据库文件！

- 推荐需要备份的目录：

由上面的介绍来看的话，如果你的硬件或者是由于经费的关系而无法全部的数据都予以备份时， 鸟哥建议你至少需要备份这些目录吻！

- /etc
- /home
- /root
- /var/spool/mail/, /var/spool/cron/, /var/spool/at/
- /var/lib/

- 不需要备份的目录：

有些数据是不需要备份的啦！例如我们在第五章文件权限与目录配置里头提到的 /proc 这个目录是在记录目前系统上面正在跑的程序，这个数据根本就不需要备份的呢！此外，外挂的机器，例如 /mnt 或 /media 里面都是挂载了其他的硬盘设备、光驱、软盘机等等，这些也不需要备份吧？ 所以啰！下面有些目录可以不需要备份啦！

- /dev ：这个随便你要不要备份
- /proc, /sys, /run：这个真的不需要备份啦！
- /mnt, /media：如果你没有在这个目录内放置你自己系统的东西，也不需要备份
- /tmp ：干嘛存暂存盘！不需要备份！

20.3.3 备份用储存媒体的选择

用来储存备份数据的媒体非常的多样化，那该如何选择呢？在选择之前我们先来讲个小故事先！

- 一个实际发生的故事

在备份的时候，选择一个“数据存放的地方”也是很需要考虑的一个因素！ 什么叫做数据存放的地方呢？讲个最简单的例子好了，我们知道说，较为大型的机器都会使用 tape 这一种磁带机来备份数据，早期如果是一般个人计算机的话，很可能是使用类似 Mo 这一种可读写式光盘片来存取数据！近来因为 USB 界面的大容量磁盘机越来越便宜且速度越来越快， 所以几乎取代了上述的总总储存媒体了！但是你不要忘记了几个重要的因素，那就是万一你的 Linux 主机被偷了呢？

这不是不可能的，之前鸟哥在成大念书时 （2000年前后），隔壁校区的研究室曾经遭小偷，里面所有的计算机都被偷走了！包括“Mo 片”，当他们发现的时候，一开始以为是硬件被偷走了，还好， 他们都有习惯进行备份，但是很不幸的，这一次连“备份的 MO 都被拿走了！”怎么办？！只能道德劝说小偷先生能够良心发现的将硬盘拿回来啰！唉～真惨....

- 异地备援系统

这个时候，所谓的“异地备援系统”就显的相当的重要了！ 什么是异地备援呀！说的太文言了！呵！简单的说，就是将你的系统数据“备份”到其他的地方去， 例如说我的机器在台南，但是我还有另一部机器在高雄老家，这样的话， 我可以将台南机器上面重要的数据都给他定期的自动的通过网络传输回去！ 也可以将家里重要的数据给他丢到台南来！这样的最大优点是可以在台南的机器死掉的时候， 即使是遭小偷，也可以有一个“万一”的备份所在！

有没有缺点啊？有啊！缺点就是～带宽严重的不足！ 在这种状态下，所能采取的策略大概就是“仅将最重要的数据给他传输回去啰！” 至于一些只要系统从新安装就可以回复的咚咚！那就没有这个必要了！当然啰，如果你的网络是属于双向 100Mbps 或 300Mbps 那就另当一回事， 想完整备份将数据丢到另一地去，也是很可行的啦！只是鸟哥没有那么好命...住家附近连 100/40 Mbps 的网络带宽都没有...

- 储存媒体的考虑

在此同时，我们再来谈一谈，那么除了异地备援这个“相对较为安全的备份”方法之外， 还有没有其他的方法可以储存备份的呢？毕竟这种网络备援系统实在是太耗带宽了！那么怎么办？喔～那就只好使用近端的设备来备份啰！ 这也是目前我们最常见到的备份方法！

在过去我们使用的储存媒体可能有 Tape, Mo, Zip, CD-RW, DVD-RW, 外接式磁盘等等， 近年来由于磁盘容量不断上提，加上已经有便宜的桌上型 NAS 储存设备，这些 NAS 储存设备就等于是一部小型 Linux server， 里面还能够提供客制化的服务，包括不同的连接界面与传输协定，因此，你只要记得，就是买还能够自我容错的 NAS 设备来备份就对了！

在经费充足的情况考虑之下，鸟哥相当建议您使用外接式的 NAS 设备，所谓的 NAS 其实就是一台内嵌 Linux 或 unix-like 的小型服务器，可能提供硬件或软件的磁盘阵列，让你可以架设 RAID10 或 RAID5,6 等的等级，所以 NAS 本身的数据就已经有保障！ 然后跟你预计要备份的 Linux server 通过网络连线，你的数据就可以直接传输到 NAS 上头去了！其他以前需要考虑的注意事项， 几乎都不再有限制～最多就是担心 NAS 的硬件坏掉而已～

若经费不足怎办，现在随便磁盘都有 4TB 以上的容量，拿一颗磁盘通过外接式 USB 界面，搭配 USB 3.0 来传输～ 随便都能够进行备份了！虽然这样的处理方式最怕的是单颗磁盘损毁，不过，如果担心的话，买两三颗来互相轮流备份，也能够处理掉这个问题！ 因为目前的数据量越来越大，实在没啥意义再使用类似 DVD 之类的储存设备来备份了！

如果你想要有比较长时间的备份储存，同时也比较担心碰撞的问题，目前企业界还是很多人会喜欢使用 Tape 来储存就是了！ 不过听业界的朋友说，磁带就是比较怕被消磁以及发霉的问题～否则，这家伙倒是很受企业备份的喜好需求！

20.4 备份的种类、频率与工具的选择

讲了好多口水了，还是没有讲到重点，真是的....好了，再来提到那个备份的种类，因为想要选择什么储存媒体与相关备份工具， 都与备份使用的方式有关！那么备份有哪些方式呢？一般可以粗略分为“累积备份”与“差异备份”这两种 [1]。当然啦，如果你在系统出错时想要重新安装到更新的系统时，仅备份关键数据也就可以了！

20.4.1 完整备份之累积备份 （Incremental backup）

备份不就是将重要数据复制出来即可吗？干嘛需要完整备份 （Full backup） 呢？如果你的主机是负责相当重要的服务， 因此如果有不明原因的死机事件造成系统损毁时，你希望在最短的时间内复原系统。此时，如果仅备份关键数据时， 那么你得要在系统出错后，再去找新的 Linux distribution 来安装，安装完毕后还得要考虑到数据新旧版本的差异问题， 还得要进行数据的移植与系统服务的重新创建等等，等到创建妥当后，还得要进行相关测试！ 这种种的工作可至少得要花上一个星期以上的工作天才能够处理妥当！所以，仅有关键数据是不够的！

- 还原的考虑

但反过来讲，如果是完整备份的话呢？若硬件出问题导致系统损毁时，只要将完整备份拿出来，整个给他倾倒回去硬盘， 所有事情就搞定了！有些时候 （例如使用 dd 指令） 甚至连系统都不需要重新安装！反正整个系统都给他倒回去，连同重要的 Linux 系统文件等，所以当然也就不需要重新安装啊！因此，很多企业用来提供重要服务的主机都会使用完整备份， 若所提供的服务真的非常重要时，甚至会再架设一部一模一样的机器呢！如此一来， 若是原本的机器出问题，那就立刻将备份的机器拿出来接管！以使企业的网络服务不会中断哩！

那你知道完整备份的定义了吧？没错！完整备份就是将根目录 （/） 整个系统通通备份下来的意思！ 不过，在某些场合下面，完整备份也可以是备份一个文件系统 （filesystem）！例如 /dev/sda1 或 /dev/md0 或 /dev/myvg/mylv 之类的文件系统就是了。

- 累积备份的原则

虽然完整备份在还原方面有相当良好的表现，但是我们都知道系统用的越久，数据量就会越大！如此一来， 完整备份所需要花费的时间与储存媒体的使用就会相当麻烦～所以，完整备份并不会也不太可能每天都进行的！ 那你想要每天都备份数据该如何进行呢？有两种方式啦，一种是本小节会谈到的累积备份，一种则是下个小节谈到的差异备份。

所谓的累积备份，指的是在系统在进行完第一次完整备份后，经过一段时间的运行， 比较系统与备份文件之间的差异，仅备份有差异的文件而已。而第二次累积备份则与第一次累积备份的数据比较， 也是仅备份有差异的数据而已。如此一来，由于仅备份有差异的数据，因此备份的数据量小且快速！备份也很有效率。 我们可以从下图来说明：

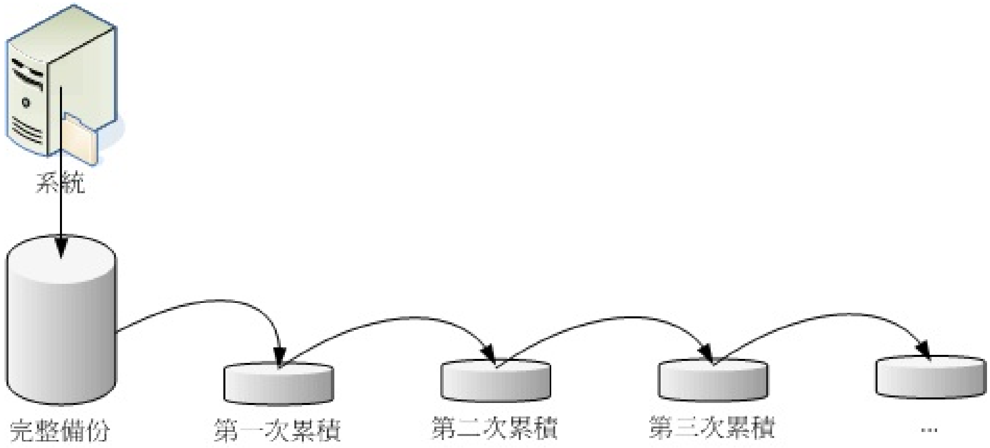


图20.4.1、累积备份（incremental backup）操作示意图

假如我在星期一作好完整备份，则星期二的累积备份是系统与完整备份间的差异数据；星期三的备份是系统与星期二的差异数据， 星期四的备份则是系统与星期三的差异数据。那你得要注意的是，星期二的数据是完整备份加第一次累积备份， 星期三的数据是完整备份加第一次累积与第二次累积备份，星期四的数据则是星期一的完整备份加第一次加第二次加第三次累积备份。 由于每次都仅与前一次的备份数据比较而已，因此备份的数据量就会少很多！

那如何还原？经过上面的分析，我们也会知道累积备份的还原方面比较麻烦！ 假设你的系统在星期五的时候挂点了！那你要如何还原？首先，你必须要还原星期一的完整备份，然后还原星期二的累积备份， 再依序还原星期三、星期四的累积备份才算完全复原！那如果你是经过了九次的累积备份，就得要还原到第九次的阶段， 才是最完整的还原程序！

- 累积备份使用的备份软件

完整备份常用的工具有 dd, cpio, xfsdump/xfsrestore 等等。因为这些工具都能够备份设备与特殊文件！ dd 可以直接读取磁盘的扇区 （sector） 而不理会文件系统，是相当良好的备份工具！不过缺点就是慢很多！ cpio 是能够备份所有文件名，不过，得要配合 find 或其他找文件名的指令才能够处理妥当。以上两个都能够进行完整备份， 但累积备份就得要额外使用脚本程序来处理。可以直接进行累积备份的就是 xfsdump 这个指令啰！详细的指令与参数用法， 请前往第八章查阅，这里仅列出几个简单的范例而已。

```
# 1\. 用 dd 来将 /dev/sda 备份到完全一模一样的 /dev/sdb 硬盘上：
[root@study ~]# dd if=/dev/sda of=/dev/sdb
# 由于 dd 是读取扇区，所以 /dev/sdb 这颗磁盘可以不必格式化！非常的方便！
# 只是你会等非常非常久！因为 dd 的速度比较慢！

# 2\. 使用 cpio 来备份与还原整个系统，假设储存媒体为 SATA 磁带机：
[root@study ~]# find / -print &#124; cpio -covB &gt; /dev/st0  &lt;==备份到磁带机
[root@study ~]# cpio -iduv &lt; /dev/st0                   &lt;==还原
```

假设 /home 为一个独立的文件系统，而 /backupdata 也是一个独立的用来备份的文件系统，那如何使用 dump 将 /home 完整的备份到 /backupdata 上呢？可以像下面这样进行看看：

```
# 1\. 完整备份
[root@study ~]# xfsdump -l 0 -L 'full' -M 'full' -f /backupdata/home.dump /home

# 2\. 第一次进行累积备份
[root@study ~]# xfsdump -l 1 -L 'full-1' -M 'full-1' -f /backupdata/home.dump1 /home
```

除了这些指令之外，其实 tar 也可以用来进行完整备份啦！举例来说，/backupdata 是个独立的文件系统， 你想要将整个系统通通备份起来时，可以这样考虑：将不必要的 /proc, /mnt, /tmp 等目录不备份，其他的数据则予以备份：

```
[root@study ~]# tar --exclude /proc --exclude /mnt --exclude /tmp \
&gt; --exclude /backupdata -jcvp -f /backupdata/system.tar.bz2 /
```

20.4.2 完整备份之差异备份 （Differential backup）

差异备份与累积备份有点类似，也是需要进行第一次的完整备份后才能够进行。只是差异备份指的是：每次的备份都是与原始的完整备份比较的结果。所以系统运行的越久，离完整备份时间越长， 那么该次的差异备份数据可能就会越大！差异备份的示意图如下所示：

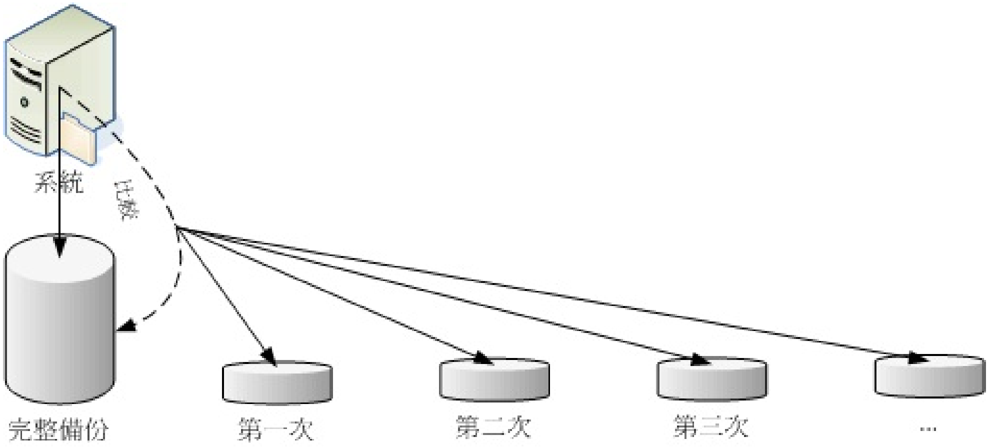


图20.4.2、差异备份（differential backup） 操作示意图

差异备份常用的工具与累积备份差不多！因为都需要完整备份嘛！如果使用 xfsdump 来备份的话，那么每次备份的等级 （level） 就都会是 level 1 的意思啦！当然啦，你也可以通过 tar 的 -N 选项来备份喔！如下所示：

```
[root@study ~]# tar -N '2015-09-01' -jpcv -f /backupdata/home.tar.bz2 /home
# 只有在比 2015-09-01 还要新的文件，在 /home 下面的文件才会被打包进 home.bz2 中！
# 有点奇怪的是，目录还是会被记录下来，只是目录内的旧文件就不会备份。
```

此外，你也可以通过 rsync 来进行镜像备份喔！ 这个 rsync 可以对两个目录进行镜像 （mirror） ，算是一个非常快速的备份工具！简单的指令语法为：

```
[root@study ~]# rsync -av 来源目录 目标目录

# 1\. 将 /home/ 镜像到 /backupdata/home/ 去
[root@study ~]# rsync -av /home /backupdata/
# 此时会在 /backupdata 下面产生 home 这个目录来！
[root@study ~]# rsync -av /home /backupdata/
# 再次进行会快很多！如果数据没有更动，几乎不会进行任何动作！
```

根据分析 [2] ，差异备份所使用的磁盘容量可能会比累积备份来的大，但是差异备份的还原较快， 因为只需要还原完整备份与最近一次的差异备份即可。无论如何，请依据你自己的喜好来选择备份的方式吧！

20.4.3 关键数据备份

完整备份虽然有许多好处，但就是需要花费很多时间！所以，如果在主机提供的服务并不是一定要 24 小时提供的前提下， 我们可以仅备份重要的关键数据即可。由于主机即使死机个一两天可能也不会影响到你的正常生活时， 仅备份关键数据就好啦！不需要整个系统都备份。仅备份关键数据是有许多好处的！ 由于完整备份可能是在系统运行期间进行，不但会花费非常多时间，而且如果备份当时系统已经被攻破， 那你备份的数据是有问题的，那还原回去也是有问题的系统啊！

如果仅是备份关键数据而已，那么由于系统的绝大部分可执行文件都可以后来重新安装，因此若你的系统不是因为硬件问题， 而是因为软件问题而导致系统被攻破或损毁时，直接捉取最新的 Linux distribution ，然后重新安装， 然后再将系统数据 （如帐号/密码与主文件夹等等） 与服务数据 （如 www/email/crontab/ftp 等等） 一个一个的填回去！ 那你的系统不但保持在最新的状态，同时也可以趁机处理一下与重新温习一下系统设置！是很不错的呦！

不过，备份关键数据最麻烦的地方其实就是在还原啦！上述的还原方式是你必须要很熟悉系统运行， 否则还原得要花费很多时间的！尤其近来的 Linux 强调安全性，所以加入 SELinux 了，你如果要从旧版的 Linux 升级到新版时， 原本若没有 SELinux 而换成新版则需要启动 SELinux 时，那个除错的时间会花很长一段日子哩！ 鸟哥认为这是仅备份关键数据的一些优缺点啦～

备份关键数据鸟哥最爱使用 tar 来处理了！如果想要分门别类的将各种不同的服务在不同的时间备份使用不同文件名， 配合 date 指令是非常好用的工具！例如下面的案例是依据日期来备份 mariadb 的数据库喔！

```
[root@study ~]# tar -jpcvf mysql.`date +%Y-%m-%d`.tar.bz2 /var/lib/mysql
```

备份是非常重要的工作，你可不希望想到才进行吧？交给系统自动处理就对啦！请自己撰写 script ， 配合 crontab 去执行吧！这样子，备份会很轻松喔！



Tips 事实上除了这些基本的 Linux 备份还原工具之外，如果你还想要尝试裸机复原的功能，那可以使用台湾国家高速网络中心开发的再生龙软件！ 这个软件相当棒！鸟哥目前服务的单位也是通过这个软件来处理整间计算机教室的复原工作喔！这个软件也有单机版， 也挺好用的！有兴趣的朋友得要自行处理软件的使用喔：

- http://clonezilla.nchc.org.tw/

20.5 鸟哥的备份策略

每部主机的任务都不相同，重要的数据也不相同，重要性也不一样，因此，每个人的备份思考角度都不一样！ 有些备份策略是非常有趣的，包括使用多个磁带机与磁带来自动备份企业数据哩 [3] 。

就鸟哥的想法来说，鸟哥并没有想要将整个系统完整的备份下来，因为太耗时间了！而且就鸟哥的立场而言，似乎也没有这个必要， 所以通常鸟哥只备份较为重要的文件而已！不过，由于鸟哥需要备份 /home 与网页数据，如果天天都备份，我想，系统迟早会受不了 （因为这两个部分就已经占去数 10 GB 的硬盘空间...），所以鸟哥就将我的备份分为两大部分，一个是每日备份经常性变动的重要数据， 一个则是每周备份就不常变动的信息。这个时候我就写了两个简单的 scripts ，分别来储存这些数据。

所以针对鸟哥的“鸟站”来说，我的备份策略是这样的：

1. 主机硬件：使用一个独立的 filesystem 来储存备份数据，此 filesystem 挂载到 /backup 当中；
2. 每日进行：目前仅备份 MySQL 数据库；
3. 每周进行：包括 /home, /var, /etc, /boot, /usr/local 等目录与特殊服务的目录；
4. 自动处理：这方面利用 /etc/crontab 来自动提供备份的进行；
5. 异地备援：每月定期的将数据分别 （a）烧录到光盘上面 （b）使用网络传输到另一部机器上面。

那就来看看鸟哥是怎么备份的吧！ ^_^

20.5.1 每周系统备份的 script

下面提供鸟哥的备份的 scripts ，希望对大家有点帮助！鸟哥假设你已经知道如何挂载一个新的 filesystem 到 /backup 去，所以格式化与挂载这里就不再强调啰。

```
[root@study ~]# vi /backup/backupwk.sh
#!/bin/bash
# ====================================================================
# 使用者参数输入位置：
# basedir=你用来储存此脚本所预计备份的数据之目录（请独立文件系统）
basedir=/backup/weekly  &lt;==您只要改这里就好了！

# ====================================================================
# 下面请不要修改了！用默认值即可！
PATH=/bin:/usr/bin:/sbin:/usr/sbin; export PATH
export LANG=C

# 设置要备份的服务的配置文件，以及备份的目录
named=$basedir/named
postfixd=$basedir/postfix
vsftpd=$basedir/vsftp
sshd=$basedir/ssh
sambad=$basedir/samba
wwwd=$basedir/www
others=$basedir/others
userinfod=$basedir/userinfo
# 判断目录是否存在，若不存在则予以创建。
for dirs in $named $postfixd $vsftpd $sshd $sambad $wwwd $others $userinfod
do
    [ ! -d "$dirs" ] && mkdir -p $dirs
done

# 1\. 将系统主要的服务之配置文件分别备份下来，同时也备份 /etc 全部。
cp -a /var/named/chroot/{etc,var}    $named
cp -a /etc/postfix /etc/dovecot.conf    $postfixd
cp -a /etc/vsftpd/*            $vsftpd
cp -a /etc/ssh/*            $sshd
cp -a /etc/samba/*            $sambad
cp -a /etc/{my.cnf,php.ini,httpd}    $wwwd
cd /var/lib
  tar -jpc -f $wwwd/mysql.tar.bz2     mysql
cd /var/www
  tar -jpc -f $wwwd/html.tar.bz2     html cgi-bin
cd /
  tar -jpc -f $others/etc.tar.bz2    etc
cd /usr/
  tar -jpc -f $others/local.tar.bz2    local

# 2\. 关于使用者参数方面
cp -a /etc/{passwd,shadow,group}    $userinfod
cd /var/spool
  tar -jpc -f $userinfod/mail.tar.bz2    mail
cd /
  tar -jpc -f $userinfod/home.tar.bz2    home
cd /var/spool
  tar -jpc -f $userinfod/cron.tar.bz2    cron at

[root@study ~]# chmod 700 /backup/backupwk.sh
[root@study ~]# /backup/backupwk.sh  &lt;==记得自己试跑看看！
```

上面的 script 主要均使用 CentOS 7.x （理论上， Red Hat 系列的 Linux 都适用） 默认的服务与目录， 如果你有设置某些服务的数据在不同的目录时，那么上面的 script 是还需要修改的！不要只是拿来用而已喔！ 上面 script 可以在下面的链接取得。

- http://linux.vbird.org/linux_basic/0580backup/backupwk-0.1.sh

20.5.2 每日备份数据的 script

再来，继续提供一下每日备份数据的脚本程序！请注意，鸟哥这里仅有提供 Mariadb 的数据库备份目录， 与 WWW 的类似留言版程序使用的 CGI 程序与写入的数据而已。 如果你还有其他的数据需要每日备份，请自行照样造句啰！ ^_^

```
[root@study ~]# vi /backup/backupday.sh
#!/bin/bash
# =========================================================
# 请输入，你想让备份数据放置到那个独立的目录去
basedir=/backup/daily/  &lt;==你只要改这里就可以了！

# =========================================================
PATH=/bin:/usr/bin:/sbin:/usr/sbin; export PATH
export LANG=C
basefile1=$basedir/mysql.$（date +%Y-%m-%d）.tar.bz2
basefile2=$basedir/cgi-bin.$（date +%Y-%m-%d）.tar.bz2
[ ! -d "$basedir" ] && mkdir $basedir

# 1\. MysQL （数据库目录在 /var/lib/mysql）
cd /var/lib
  tar -jpc -f $basefile1 mysql

# 2\. WWW 的 CGI 程序 （如果有使用 CGI 程序的话）
cd /var/www
  tar -jpc -f $basefile2 cgi-bin

[root@study ~]# chmod 700 /backup/backupday.sh
[root@study ~]# /backup/backupday.sh  &lt;==记得自己试跑看看！
```

上面的脚本可以在下面的链接取得。这样一来每天的 Mariadb 数据库就可以自动的被记录在 /backup/daily/ 目录里头啦！而且还是文件名称会自动改变的呦！呵呵！我很喜欢！OK！再来就是开始让系统自己跑啦！ 怎么跑？就是 /etc/crontab 呀！提供一下我的相关设置呦！

- http://linux.vbird.org/linux_basic/0580backup/backupday.sh

```
[root@study ~]# vi /etc/crontab
# 加入这两行即可 （请注意你的文件目录！不要照抄呦！）
30 3 * * 0 root /backup/backupwk.sh
30 2 * * * root /backup/backupday.sh
```

这样系统就会自动的在每天的 2:30 进行 Mariadb 的备份，而在每个星期日的 3:30 进行重要文件的备份！呵呵！你说，是不是很容易呢！但是请千万记得呦！还要将 /backup/ 当中的数据 copy 出来才行耶！否则整部系统死掉的时候...那可不是闹着玩的！ 所以鸟哥大约一个月到两个月之间，会将 /backup 目录内的数据使用 DVD 复制一下，然后将 DVD 放置在家中保存！这个 DVD 很重要的喔！不可以遗失，否则系统的重要数据 （尤其是帐号信息） 流出去可不是闹着玩的！



Tips 有些时候，你在进行备份时，被备份的文件可能同时间被其他的网络服务所修改喔！ 举例来说，当你备份 Mariadb 数据库时，刚好有人利用你的数据库发表文章，此时， 可能会发生一些错误的讯息。要避免这类的问题时，可以在备份前，将该服务先关掉， 备份完成后，再启动该服务即可！感谢讨论区 duncanlo 提供这个方法！

20.5.3 远端备援的 script

如果你有控管两部以上的 Linux 主机时，那么互相将对方的重要数据保存一份在自己的系统中也是个不错的想法！ 那怎么保存啊？使用 USB 复制来去吗？当然不是啦！你可以通过网络来处置啦！我们假设你已经有一部主机， 这部主机的 IP 是 192.168.1.100 ，而且这部主机已经提供了 sshd 这个网络服务了，接下来你可以这样作：

- 使用 rsync 上传备份数据

要使用 rsync 你必须要在你的服务器上面取得某个帐号使用权后， 并让该帐号可以不用密码即可登陆才行！这部分得要先参考服务器篇的远端连线服务器才行！ 假设你已经设置好 dmtsai 这个帐号可以不用密码即可登陆远端服务器，而同样的你要让 /backup/weekly/ 整个备份到 /home/backup/weekly 下面时，可以简单这样做：

```
[root@study ~]# vi /backup/rsync.sh
#!/bin/bash
remotedir=/home/backup/
basedir=/backup/weekly
host=127.0.0.1
id=dmtsai

# 下面为程序阶段！不需要修改喔！
rsync -av -e ssh $basedir ${id}@${host}:${remotedir}
```

由于 rsync 可以通过 ssh 来进行镜像备份，所以没有变更的文件将不需要上传的！相当的好用呢！ 好了！大家赶紧写一个适合自己的备份 script 来进行备份的行为吧！重要重要喔！



Tips 因为 rsync 搭配 sshd 真的很好用！加上它本身就有加密～近期以来大家对于数据在网络上面跑都非常的在乎安全性， 所以鸟哥就取消了 FTP 的传输方式啰～

20.6 灾难复原的考虑

之所以要备份当然就是预防系统挂点啦！如果系统真的挂点的话，那么你该如何还原系统呢？

- 硬件损毁，且具有完整备份的数据时

由于是硬件损毁，所以我们不需要考虑系统软件的不稳定问题，所以可以直接将完整的系统复原回去即可。 首先，你必须要先处理好你的硬件，举例来说，将你的硬盘作个适当的处理，譬如创建成为磁盘阵列之类的。 然后依据你的备份状态来复原。举例来说，如果是使用差异备份，那么将完整备份复原后， 将最后一次的差异备份复原回去，你的系统就恢复了！非常简单吧！

- 由于软件的问题产生的被攻破资安事件

由于系统的损毁是因为被攻击，此时即使你恢复到正常的系统，那么这个系统既然会被攻破， 没道理你还原成旧系统就不会被再次攻破！所以，此时完整备份的复原可能不是个好方式喔！最好是需要这样进行啦：

1. 先拔除网络线，最好将系统进行完整备份到其他媒体上，以备未来查验
2. 开始查阅登录文件，尝试找出各种可能的问题
3. 开始安装新系统 （最好找最新的 distribution）
4. 进行系统的升级，与防火墙相关机制的制订
5. 根据 2 的错误，在安装完成新系统后，将那些 bug 修复
6. 进行各项服务与相关数据的恢复
7. 正式上线提供服务，并且开始测试

软件资安事件造成的问题可大可小，一般来说，标准流程都是建议你将出问题的系统备份下来， 如果被追踪到你的主机曾经攻击过别人的话，那么你至少可以拿出备份数据来佐证说，你是被攻击者， 而不是主动攻击别人的坏人啊！然后，记得一定要找出问题点并予以克服，不然的话，你的系统将一再地被攻击啊！ 那样可就伤脑筋啰～

20.7 重点回顾

- 网际网络（Internet）就是 TCP/IP ，而 IP 的取得需与 ISP 要求。一般常见的取得 IP 的方法有：（1）手动直接设置（2）自动取得（dhcp）（3）拨接取得（4）cable宽带等方式。
- 主机的网络设置要成功，必须要有下面的数据：（1）IP（2）Netmask（3）gateway（4）DNS 服务器 等项目；
- 本章新增硬件信息的收集指令有： lspci, lsusb, iostat 等；
- 备份是系统损毁时等待救援的救星，但造成系统损毁的因素可能有硬件与软件等原因。
- 由于主机的任务不同，备份的数据与频率等考虑参数也不相同。
- 常见的备份考虑因素有：关键文件、储存媒体、备份方式（完整/关键）、备份频率、使用的备份工具等。
- 常见的关键数据有：/etc, /home, /var/spool/mail, /boot, /root 等等
- 储存媒体的选择方式，需要考虑的地方有：备份速度、媒体的容量、经费与媒体的可靠性等。
- 与完整备份有关的备份策略主要有：累积备份与差异备份。
- 累积备份可具有较小的储存数据量、备份速度快速等。但是在还原方面则比差异备份的还原慢。
- 完整备份的策略中，常用的工具有 dd, cpio, tar, xfsdump 等等。

20.8 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看） 简答题部分：

- 如果你想要知道整个系统的周边硬件设备，可以使用哪个指令查询？lspci 可以查询到，更可使用 lspci -v 来查询更详细信息。
- 承上题，那么如果单纯只想要知道 USB 设备呢？又该如何查询？lsusb就可以查询的到！
- （挑战题）如果你的网络设置妥当了，但是却老是发现网络不通，你觉得应该如何进行测试？（1）先检查硬件，每个环节 （网卡、hub/switch、路由器等） 的灯号是否有亮？有亮再进行下个动作；（2）使用 ifconfig 检查 IP 与 netmask 的数据是否正确，若正确才可进行下一步；（3）使用 route 看看 default gateway 是否正确，若正确再进行下一步；（4）使用 ping -c 3 [gateway IP] ，若有回应才进行下一步；（5）使用 ping -c 3 [外部IP，例如 168.95.1.1] ，若有回应则 IP 正常，若无回应，请检查 gateway 的设置（6）使用 dig www.google.com 看看能否找到 IP ，找不到则请检查 /etc/resolv.conf 的设置。
- 挑战题：尝试将你在学习本书所进行的各项任务备份下来，然后删除你的系统，接下来重新安装最新的 CentOS 7.x ， 再将你备份的数据复原回来，看看能否成功的让你的系统回复到之前的状态呢？
- 挑战题：查询一下何谓企鹅龙软件，讨论一下该软件的还原机制是属于累积备份？还是完整备份？
- 常用的完整备份 （full backup） 工具指令有哪些？xfsdump + xfsrestore, dd, cpio 搭配 find 等软件。
- 你所看到的常见的储存设备有哪些？Floppy, Mo, Zip, CD-RW, DVD-RW, 外接式 USB 硬盘, Tape, 外接式储存阵列 （RAID），额外的储存架构，如 SAN, NAS 等。

20.9 参考资料与延伸阅读

- [1]维基百科的备份说明：http://en.wikipedia.org/wiki/Incremental_backup
- [2]关于 differential 与 incremental 备份的优缺点说明：http://www.backupschedule.net/databackup/differentialbackup.html
- [3]一些备份计划的实施：http://en.wikipedia.org/wiki/Backup_rotation_scheme

2005/10/25：准备准备～写一些跟硬件比较有关系的数据！ 2005/11/08：准备完毕 USB 与 lm_sensors 的部分了～啊！拖了真久～还有 RAID 的说明也差不多哩！ 2005/11/09：加入了 FC4 的setup 指令，尤其是打印机的部分，可以参考参考！ 2005/11/10：终于将 iSCSI 的设备写好了～这部份真的是很有趣！不过，一般使用者可能碰不到就是了。 2005/11/13：终于将 CUPS 架构设置好自己的 Printer 部分了！ 2005/11/14：连同 LVM 也大致的给他写完了！那个 resize2fs 指令确实有趣！ 2005/11/25：加入一个简单的练习题～利用 dd 配合 resize2fs 来制作备份的数据！ 2009/04/30：将 LVM 移动到 第十五章 ，且拿掉 iSCSI 的说明了。 2009/04/30：将旧的基于 FC4 撰写的版本移动到 此处 。 2009/06/03：加入 udev 与 hal 的简单说明！ 2009/09/15：简单修订一些语句，修改章节的习题，并没有改到什么重要的信息。 2015/08/31：将旧的基于 CentOS 5 的版本移动到 这里。 2015/09/xx：将备份策略的文章也挪到本章来，同时移除很多数据，包括 CUPS 打印机等等都拿掉了！

第二十一章、软件安装：源代码与 Tarball

最近更新日期：20//

我们在第一章、Linux是什么当中提到了 GNU 计划与 GPL 授权所产生的自由软件与开放源码等咚咚。不过，前面的章节都还没有提到真正的开放源码是什么的讯息！在这一章当中，我们将借由 Linux 操作系统里面的可执行文件，来理解什么是可执行的程序，以及了解什么是编译器。另外，与程序息息相关的函数库（library）的信息也需要了解一番！不过，在这个章节当中，鸟哥并不是要你成为一个开放源码的程序设计师， 而是希望你可以了解如何将开放源码的程序设计、加入函数库的原理、通过编译而成为可以执行 的 binary program，最后该可执行文件可被我们所使用的一连串过程！

了解上面的咚咚有什么好处呢？因为在 Linux 的世界里面，由于客制化的关系，有时候我们需要自行安装软件在自己的 Linux 系统上面，所以如果你有简单的程序编译概念，那么将很容易进行软件的安装。 甚至在发生软件编译过程中的错误时，你也可以自行作一些简易的修订呢！而最传统的软件安装过程， 自然就是由源代码编译而来的啰！所以，在这里我们将介绍最原始的软件管理方式：使用 Tarball 来安装与升级管理我们的软件喔！

20.1 开放源码的软件安装与升级简介

如果鸟哥想要在我的 Linux 服务器上面跑网页服务器 （WWW server） 这项服务，那么我应该要做些什么事呢？当然就一定需要“安装网页服务器的软件”啰！如果鸟哥的服务器上面没有这个软件的话，那当然也就无法启用 WWW 的服务啦！所以啦，想要在你的 Linux 上面进行一些有的没的功能，学会“如何安装软件”是很重要的一个课题！

咦！安装软件有什么难的？在 W 牌的操作系统上面安装软件时，不是只要一直给他按 “下一步”就可以安装妥当了吗？话是这样说没错啦，不过，也由于如此，所以在 Windows 系统上面的软件都是一模一样的，也就是说，你“无法修改该软件的原始程序码”，因此， 万一你想要增加或者减少该软件的某些功能时，大概只能求助于当初发行该软件的厂商了！（这就是所谓的商机吗？）

或许你会说：“唉呦！我不过是一般人，不会用到多余的功能，所以不太可能会更动到程序码的部分吧？” 如果你这么想的话，很抱歉～是有问题的！怎么说呢？像目前网络上面的病毒、黑客软件、臭虫程序等等， 都可能对你的主机上面的某些软件造成影响，导致主机的死机或者是其他数据损毁等等的伤害。 如果你可以借由安全信息单位所提供的修订方式进行修改， 那么你将可以很快速的自行修补好该软件的漏洞，而不必一定要等到软件开发商提供修补的程序包哩！要知道，提早补洞是很重要的一件事。



Tips 并不是软件开发商故意要搞出一个有问题的软件，而是某些程序码当初设计时可能没有考虑周全， 或者是程序码与操作系统的权限设置并不相同，所导致的一些漏洞。当然，也有可能是 cracker 通过某些攻击程序测试到程序的不周全所致。 无论如何，只要有网络存在的一天，可以想像的到，程序的漏洞永远补不完！但能补多少就补多少吧！

这样说可以了解 Linux 的优点了吗？没错！因为 Linux 上面的软件几乎都是经过 GPL 的授权，所以每个软件几乎均提供原始程序码， 并且你可以自行修改该程序码，以符合你个人的需求呢！很棒吧！这就是开放源码的优点啰！不过，到底什么是开放源码？ 这些程序码是什么咚咚？又 Linux 上面可以执行的相关软件文件与开放源码之间是如何转换的？不同版本的 Linux 之间能不能使用同一个可执行文件？或者是该可执行文件需要由原始程序码的部分重新进行转换？ 这些都是需要厘清观念的。下面我们先就原始程序码与可可执行文件来进行说明。

21.1.1 什么是开放源码、编译器与可可执行文件

在讨论程序码是什么之前，我们先来谈论一下什么是可可执行文件？我们说过，在 Linux 系统上面，一个文件能不能被执行看的是有没有可执行的那个权限 （具有 x permission），不过，Linux 系统上真正认识的可可执行文件其实是二进制文件 （ binary program），例如 /usr/bin/passwd, /bin/touch 这些个文件即为二进制程序码。

或许你会说 shell scripts 不是也可以执行吗？其实 shell scripts 只是利用 shell （例如 bash） 这支程序的功能进行一些判断式，而最终执行的除了 bash 提供的功能外，仍是调用一些已经编译好的二进制程序来执行的呢！ 当然啦， bash 本身也是一支二进制程序啊！那么我怎么知道一个文件是否为 binary 呢？还记得我们在第六章里面提到的 file 这个指令的功能吗？对啦！用他就是了！我们现在来测试一下：

```
# 先以系统的文件测试看看：
[root@study ~]# file /bin/bash
/bin/bash: ELF 64-bit LSB executable, x86-64, version 1 （SYSV）, dynamically linked
  （uses shared libs）, for GNU/Linux 2.6.32, BuildID[sha1]=0x7e60e35005254...stripped

# 如果是系统提供的 /etc/init.d/network 呢？
[root@study ~]# file /etc/init.d/network
/etc/init.d/network: Bourne-Again shell script, ASCII text executable
```

看到了吧！如果是 binary 而且是可以执行的时候，他就会显示可执行文件类别 （ELF 64-bit LSB executable），同时会说明是否使用动态函数库 （shared libs），而如果是一般的 script ，那他就会显示出 text executables 之类的字样！



Tips 事实上，network 的数据显示出 Bourne-Again ... 那一行，是因为你的 scripts 上面第一行有宣告 #!/bin/bash 的缘故，如果你将 script 的第一行拿掉，那么不管 /etc/init.d/network 的权限为何，他其实显示的是 ASCII 文本文件的信息喔！

既然 Linux 操作系统真正认识的其实是 binary program，那么我们是如何做出这样的一支 binary 的程序呢？首先，我们必须要写程序，用什么东西写程序？就是一般的文书处理器啊！鸟哥都喜欢使用 vim 来进行程序的撰写，写完的程序就是所谓的原始程序码啰！ 这个程序码文件其实就是一般的纯文本文件。 在完成这个源代码文件的编写之后，再来就是要将这个文件“编译”成为操作系统看的懂得 binary program 啰！而要编译自然就需要“编译器”来动作， 经过编译器的编译与链接之后，就会产生一支可以执行的 binary program 啰。

举个例子来说，在 Linux 上面最标准的程序语言为 C ，所以我使用 C 的语法进行原始程序码的书写，写完之后，以 Linux 上标准的 C 语言编译器 gcc 这支程序来编译，就可以制作一支可以执行的 binary program 啰。整个的流程有点像这样：

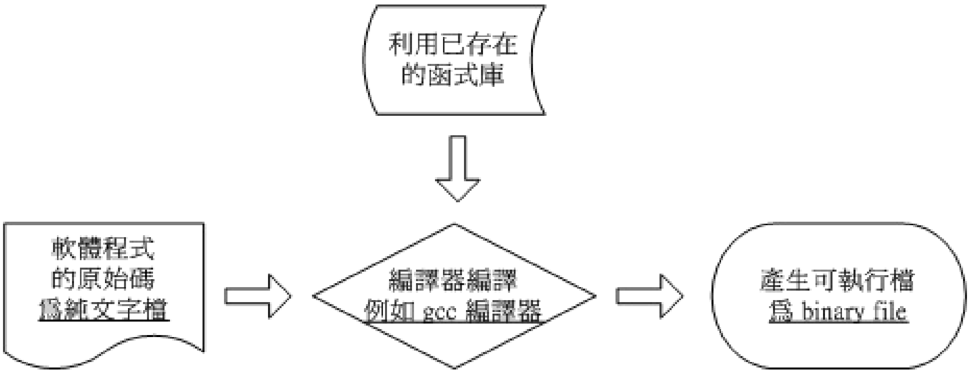


图21.1.1、利用 gcc 编译器进行程序的编译流程示意图

事实上，在编译的过程当中还会产生所谓的目标文件 （Object file），这些文件是以 *.o* 的扩展名样式存在的！至于 *C* 语言的源代码文件通常以 .c 作为扩展名。此外，有的时候，我们会在程序当中"引用、调用" 其他的外部副程序，或者是利用其他软件提供的"函数功能"，这个时候，我们就必须要在编译的过程当中， 将该函数库给他加进去，如此一来，编译器就可以将所有的程序码与函数库作一个链接 （Link） 以产生正确的可执行文件啰。

总之，我们可以这么说：

- 开放源码：就是程序码，写给人类看的程序语言，但机器并不认识，所以无法执行；
- 编译器：将程序码转译成为机器看的懂得语言，就类似翻译者的角色；
- 可可执行文件：经过编译器变成二进制程序后，机器看的懂所以可以执行的文件。

20.1.2 什么是函数库

在前一小节的图21.1.1示意图中，在编译的过程里面有提到函数库这东西。 什么是函数库呢？先举个例子来说：我们的 Linux 系统上通常已经提供一个可以进行身份验证的模块， 就是在第十三章提到的 PAM 模块。这个 PAM 提供的功能可以让很多的程序在被执行的时候，除了可以验证使用者登陆的信息外， 还可以将身份确认的数据记录在登录文件里面，以方便系统管理员的追踪！

既然有这么好用的功能，那如果我要编写具有身份认证功能的程序时，直接引用该 PAM 的功能就好啦，如此一来，我就不需要重新设计认证机制啰！也就是说，只要在我写的程序码里面，设置去调用 PAM 的函数功能，我的程序就可以利用 Linux 原本就有的身份认证的程序咯！除此之外，其实我们的 Linux 核心也提供了相当多的函数库来给硬件开发者利用喔。

函数库又分为动态与静态函数库，这两个咚咚的分别我们在后面的小节再加以说明。 这里我们以一个简单的流程图，来示意一支有调用外部函数库的程序的执行情况。

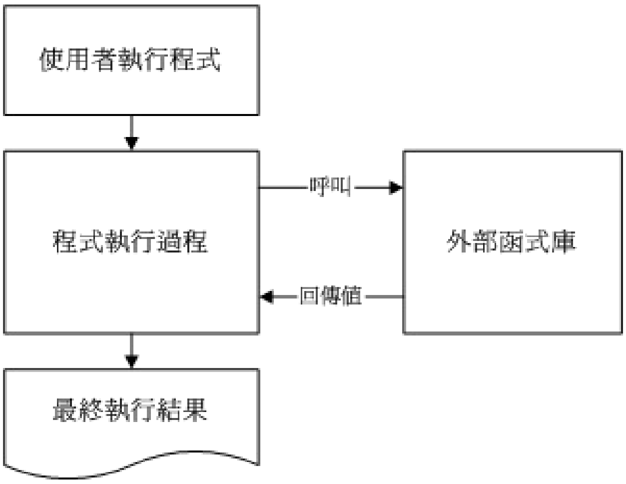


图21.1.2、程序执行时引用外部动态函数库的示意图

很简单的示意图啊！^_^！而如果要在程序里面加入引用的函数库，就需要如图 21.1.1 所示，亦即在编译的过程当中，就需要加入函数库的相关设置啰。 事实上， Linux 的核心提供很多的核心相关函数库与外部参数， 这些核心功能在设计硬件的驱动程序的时候是相当有用的信息，这些核心相关信息大多放置在 /usr/include, /usr/lib, /usr/lib64 里面哩！我们在本章的后续小节再来探讨。反正我们可以简单的这么想：

- 函数库：就类似副程序的角色，可以被调用来执行的一段功能函数。

20.1.3 什么是 make 与 configure

事实上，使用类似 gcc 的编译器来进行编译的过程并不简单，因为一套软件并不会仅有一支程序， 而是有一堆程序码文件。所以除了每个主程序与副程序均需要写上一笔编译过程的指令外，还需要写上最终的链接程序。 程序码小的时候还好，如果是类似 WWW 服务器软件（例如 Apache） ，或者是类似核心的源代码，动则数百 MBytes 的数据量，编译指令会写到疯掉～这个时候，我们就可以使用 make 这个指令的相关功能来进行编译过程的指令简化了！

当执行 make 时，make 会在当时的目录下搜寻 Makefile （or makefile） 这个文本文件，而 Makefile 里面则记录了源代码如何编译的详细信息！ make 会自动的判别源代码是否经过变动了，而自动更新可执行文件，是软件工程师相当好用的一个辅助工具呢！

咦！make 是一支程序，会去找 Makefile ，那 Makefile 怎么写？ 通常软件开发商都会写一支侦测程序来侦测使用者的作业环境， 以及该作业环境是否有软件开发商所需要的其他功能，该侦测程序侦测完毕后，就会主动的创建这个 Makefile 的规则文件啦！通常这支侦测程序的文件名为 configure 或者是 config 。

咦！那为什么要侦测作业环境呢？在第一章当中， 不是曾经提过其实每个 Linux distribution 都使用同样的核心吗？但你得要注意， 不同版本的核心所使用的系统调用可能不相同，而且每个软件所需要的相依的函数库也不相同， 同时，软件开发商不会仅针对 Linux 开发，而是

会针对整个 Unix-Like 做开发啊！ 所以他也必须要侦测该操作系统平台有没有提供合适的编译器才行！所以当然要侦测环境啊！ 一般来说，侦测程序会侦测的数据大约有下面这些：

- 是否有适合的编译器可以编译本软件的程序码；
- 是否已经存在本软件所需要的函数库，或其他需要的相依软件；
- 操作系统平台是否适合本软件，包括 Linux 的核心版本；
- 核心的表头定义文件 （header include） 是否存在 （驱动程序必须要的侦测）。

至于 make 与 configure 运行流程的相关性，我们可以使用下面的图示来示意一下啊！ 下图中，你要进行的任务其实只有两个，一个是执行 configure 来创建 Makefile ， 这个步骤一定要成功！成功之后再以 make 来调用所需要的数据来编译即可！非常简单！

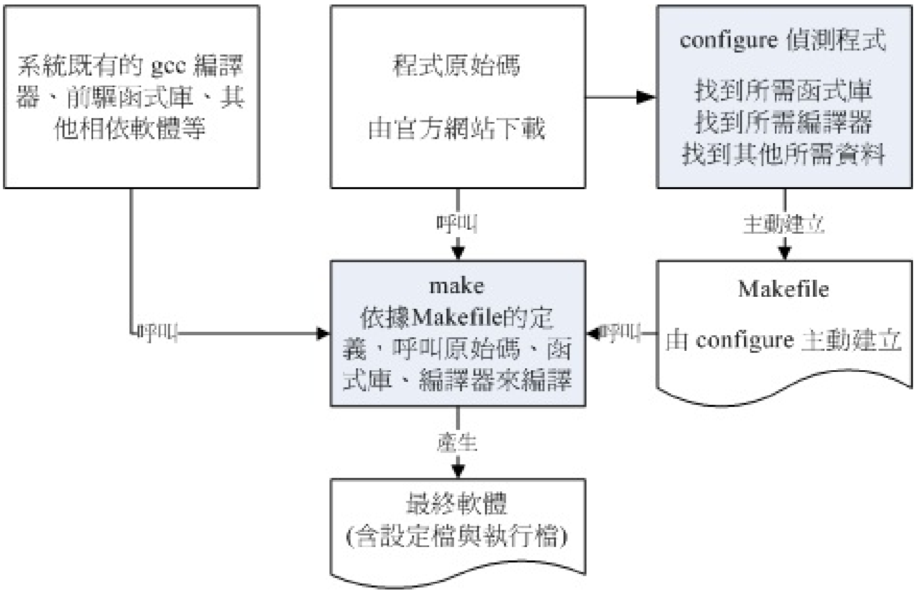


图21.1.3、通过 configure 与 make 进行编译示意图

由于不同的 Linux distribution 的函数库文件所放置的路径，或者是函数库的文件名订定， 或者是默认安装的编译器，以及核心的版本都不相同，因此理论上，你无法在 CentOS 7.x 上面编译出 binary program 后，还将他拿到 SuSE 上面执行，这个动作通常是不可能成功的！ 因为调用的目标函数库位置可能不同 （参考图21.1.2） ， 核心版本更不可能相同！所以能够执行的情况是微乎其微！所以同一套软件要在不同的平台上面执行时， 必须要重复编译！所以才需要源代码嘛！了解乎！详细的 make 用法与 Makefile 规则，在后续的小节里面再探讨啰！

20.1.4 什么是 Tarball 的软件

从前面几个小节的说明来看，我们知道所谓的原始程序码，其实就是一些写满了程序码的纯文本。 那我们在第八章压缩指令的介绍当中， 也了解了纯文本文件在网络上其实是很浪费带宽的一种文件格式！ 所以啦，如果能够将这些源代码通过文件的打包与压缩技术来将文件的数量与容量减小， 不但让使用者容易下载，软件开发商的网站带宽也能够节省很多很多啊！ 这就是 Tarball 文件的由来啰！



Tips 想一想，一个核心的源代码文件大约要 300~500 MB 以上，如果每个人都去下载这样的一个核心文件， 呵呵！那么网络带宽不被吃的死翘翘才怪呢！

所谓的 Tarball 文件，其实就是将软件的所有源代码文件先以 tar 打包，然后再以压缩技术来压缩，通常最常见的就是以 gzip 来压缩了。因为利用了 tar 与 gzip 的功能，所以 tarball 文件一般的扩展名就会写成 *.tar.gz* 或者是简写为 .tgz 啰！不过，近来由于 bzip2 与 xz 的压缩率较佳，所以 Tarball 渐渐的以 bzip2 及 xz 的压缩技术来取代 gzip 啰！因此文件名也会变成 *.tar.bz2*, .tar.xz 之类的哩。所以说， Tarball 是一个软件包， 你将他解压缩之后，里面的文件通常就会有：

- 原始程序码文件；
- 侦测程序文件 （可能是 configure 或 config 等文件名）；
- 本软件的简易说明与安装说明 （INSTALL 或 README）。

其中最重要的是那个 INSTALL 或者是 README 这两个文件，通常你只要能够参考这两个文件， Tarball 软件的安装是很简单的啦！我们在后面的章节会再继续介绍 Tarball 这个玩意儿。

20.1.5 如何安装与升级软件

将源代码作了一个简单的介绍，也知道了系统其实认识的可可执行文件是 binary program 之后，好了，得要聊一聊，那么怎么安装与升级一个 Tarball 的软件？为什么要安装一个新的软件呢？当然是因为我们的主机上面没有该软件啰！那么， 为何要升级呢？原因可能有下面这些：

- 需要新的功能，但旧有主机的旧版软件并没有，所以需要升级到新版的软件；
- 旧版本的软件上面可能有资安上的顾虑，所以需要更新到新版的软件；
- 旧版的软件执行性能不彰，或者执行的能力不能让管理者满足。

在上面的需求当中，尤其需要注意的是第二点，当一个软件有安全上的顾虑时，千万不要怀疑， 赶紧更新软件吧！否则造成网络危机，那可不是闹着玩的！那么更新的方法有哪些呢？基本上更新的方法可以分为两大类，分别是：

- 直接以源代码通过编译来安装与升级；
- 直接以编译好的 binary program 来安装与升级。

上面第一点很简单，就是直接以 Tarball 在自己的机器上面进行侦测、编译、 安装与设置等等动作来升级就是了。不过，这样的动作虽然让使用者在安装过程当中具有很高的弹性， 但毕竟是比较麻烦一点，如果 Linux distribution 厂商能够针对自己的作业平台先进行编译等过

程，再将编译好的 binary program 释出的话，那由于我的系统与该 Linux distribution 的环境是相同的，所以他所释出的 binary program 就可以在我的机器上面直接安装啦！省略了侦测与编译等等繁杂的过程呢！

这个预先编译好程序的机制存在于很多 distribution 喔，包括有 Red Hat 系统 （含 Fedora/CentOS 系列） 发展的 RPM 软件管理机制与 yum 线上更新模式； Debian 使用的 dpkg 软件管理机制与 APT 线上更新模式等等。

由于 CentOS 系统是依循标准的 Linux distribution，所以可以使用 Tarball 直接进行编译的安装与升级， 当然也可以使用 RPM 相关的机制来进行安装与升级啰！本章节主要针对 Tarball ，至于 RPM 则留待下个章节再来介绍呢！

好了，那么一个软件的 Tarball 是如何安装的呢？基本流程是这样的啦：

1. 将 Tarball 由厂商的网页下载下来；
2. 将 Tarball 解开，产生很多的源代码文件；
3. 开始以 gcc 进行源代码的编译 （会产生目标文件 object files）；
4. 然后以 gcc 进行函数库、主、副程序的链接，以形成主要的 binary file；
5. 将上述的 binary file 以及相关的配置文件安装至自己的主机上面。

上面第 3, 4 步骤当中，我们可以通过 make 这个指令的功能来简化他， 所以整个步骤其实是很简单的啦！只不过你就得需要至少有 gcc 以及 make 这两个软件在你的 Linux 系统里面才行喔！ 详细的过程以及需要的软件我们在后面的章节继续来介绍的啦！

21.2 使用传统程序语言进行编译的简单范例

经过上面的介绍之后，你应该比较清楚的知道源代码、编译器、函数库与可执行文件之间的相关性了。 不过，详细的流程可能还是不很清楚，所以，在这里我们以一个简单的程序范例来说明整个编译的过程喔！赶紧进入 Linux 系统，实地的操作一下下面的范例呢！

21.2.1 单一程序：印出 Hello World

我们以 Linux 上面最常见的 C 语言来撰写第一支程序！第一支程序最常作的就是..... 在屏幕上面印出“Hello World！”的字样～当然， 这里我们是以简单的 C 语言来撰写，如果你对于 C 有兴趣的话，那么请自行购买相关的书籍喔！ ^_^ 好了，不啰唆，立刻编辑第一支程序吧！



Tips 请先确认你的 Linux 系统里面已经安装了 gcc 了喔！如果尚未安装 gcc 的话，请先参考下一节的 RPM 安装法，先安装好 gcc 之后，再回来阅读本章。 如果你已经有网络了，那么直接使用“ yum groupinstall "Development Tools" ” 预先安装好所需的所有软件即可。 rpm 与 yum 均会在下一章介绍。

- 编辑程序码，亦即源代码

```
[root@study ~]# vim hello.c   &lt;==用 C 语言写的程序扩展名建议用 .c
#include &lt;stdio.h&gt;
int main（void）
{
        printf（"Hello World\n"）；
}
```

上面是用 C 语言的语法写成的一个程序文件。第一行的那个“ # ”并不是注解喔！如果你担心输入错误， 请到下面的链接下载这个文件：

- http://linux.vbird.org/linux_basic/0520source/hello.c

- 开始编译与测试执行

```
[root@study ~]# gcc hello.c
[root@study ~]# ll hello.c a.out
-rwxr-xr-x. 1 root root 8503 Sep  4 11:33 a.out   &lt;==此时会产生这个文件名
-rw-r--r--. 1 root root   71 Sep  4 11:32 hello.c

[root@study ~]# ./a.out
Hello World  &lt;==呵呵！成果出现了！
```

在默认的状态下，如果我们直接以 gcc 编译源代码，并且没有加上任何参数，则可执行文件的文件名会被自动设置为 a.out 这个文件名称！ 所以你就能够直接执行 ./a.out 这个可执行文件啦！上面的例子很简单吧！那个 hello.c 就是源代码，而 gcc 就是编译器，至于 a.out 就是编译成功的可执行 binary program 啰！ 咦！那如果我想要产生目标文件 （object file） 来进行其他的动作，而且可执行文件的文件名也不要用默认的 a.out ，那该如何是好？其实你可以将上面的第 2 个步骤改成这样：

```
[root@study ~]# gcc -c hello.c
[root@study ~]# ll hello*
-rw-r--r--. 1 root root   71 Sep  4 11:32 hello.c
-rw-r--r--. 1 root root 1496 Sep  4 11:34 hello.o  &lt;==就是被产生的目标文件

[root@study ~]# gcc -o hello hello.o
[root@study ~]# ll hello*
-rwxr-xr-x. 1 root root 8503 Sep  4 11:35 hello  &lt;==这就是可可执行文件！ -o 的结果
-rw-r--r--. 1 root root   71 Sep  4 11:32 hello.c
-rw-r--r--. 1 root root 1496 Sep  4 11:34 hello.o

[root@study ~]# ./hello
Hello World
```

这个步骤主要是利用 hello.o 这个目标文件制作出一个名为 hello 的可执行文件，详细的 gcc 语法我们会在后续章节中继续介绍！通过这个动作后，我们可以得到 hello 及 hello.o 两个文件， 真正可以执行的是 hello 这个 binary program 喔！ 或许你会觉得，咦！只要一个动作作出 a.out 就好了，干嘛还要先制作目标文件再做成可执行文件呢？ 呵呵！通过下个范例，你就可以知道为什么啦！

21.2.2 主、副程序链接：副程序的编译

如果我们在一个主程序里面又调用了另一个副程序呢？这是很常见的一个程序写法， 因为可以简化整个程序的易读性！在下面的例子当中，我们以 thanks.c 这个主程序去调用 thanks_2.c 这个副程序，写法很简单：

- 撰写所需要的主、副程序

```
# 1\. 编辑主程序：
[root@study ~]# vim thanks.c
#include &lt;stdio.h&gt;
int main（void）
{
        printf（"Hello World\n"）；
        thanks_2（）；
}
# 上面的 thanks_2（）； 那一行就是调用副程序啦！

[root@study ~]# vim thanks_2.c
#include &lt;stdio.h&gt;
void thanks_2（void）
{
        printf（"Thank you!\n"）；
}
```

上面这两个文件你可以到下面下载：

- http://linux.vbird.org/linux_basic/0520source/thanks.c
- http://linux.vbird.org/linux_basic/0520source/thanks_2.c

- 进行程序的编译与链接 （Link）

```
# 2\. 开始将源代码编译成为可执行的 binary file ：
[root@study ~]# gcc -c thanks.c thanks_2.c
[root@study ~]# ll thanks*
-rw-r--r--. 1 root root   75 Sep  4 11:43 thanks_2.c
-rw-r--r--. 1 root root 1496 Sep  4 11:43 thanks_2.o  &lt;==编译产生的！
-rw-r--r--. 1 root root   91 Sep  4 11:42 thanks.c
-rw-r--r--. 1 root root 1560 Sep  4 11:43 thanks.o    &lt;==编译产生的！

[root@study ~]# gcc -o thanks thanks.o thanks_2.o
[root@study ~]# ll thanks*
-rwxr-xr-x. 1 root root 8572 Sep  4 11:44 thanks    &lt;==最终结果会产生这玩意儿

# 3\. 执行一下这个文件：
[root@study ~]# ./thanks
Hello World
Thank you!
```

知道为什么要制作出目标文件了吗？由于我们的源代码文件有时并非仅只有一个文件，所以我们无法直接进行编译。 这个时候就需要先产生目标文件，然后再以链接制作成为 binary 可可执行文件。另外，如果有一天，你更新了 thanks_2.c 这个文件的内容，则你只要重新编译 thanks_2.c 来产生新的 thanks_2.o ，然后再以链接制作出新的 binary 可可执行文件即可！而不必重新编译其他没有更动过的源代码文件。 这对于软件开发者来说，是一个很重要的功能，因为有时候要将偌大的源代码全部编译完成，会花很长的一段时间呢！

此外，如果你想要让程序在执行的时候具有比较好的性能，或者是其他的除错功能时， 可以在编译的过程里面加入适当的参数，例如下面的例子：

```
[root@study ~]# gcc -O -c thanks.c thanks_2.c  &lt;== -O 为产生最优化的参数

[root@study ~]# gcc -Wall -c thanks.c thanks_2.c
thanks.c: In function ‘main’:
thanks.c:5:9: warning: implicit declaration of function ‘thanks_2’ [-Wimplicit-function-d
         thanks_2();
         ^
thanks.c:6:1: warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^
# -Wall 为产生更详细的编译过程信息。上面的讯息为警告讯息 （warning） 所以不用理会也没有关系！
```

至于更多的 gcc 额外参数功能，就得要 man gcc 啰～呵呵！可多的跟天书一样～

21.2.3 调用外部函数库：加入链接的函数库

刚刚我们都仅只是在屏幕上面印出一些字眼而已，如果说要计算数学公式呢？例如我们想要计算出三角函数里面的 sin （90度角） 。要注意的是，大多数的程序语言都是使用径度而不是一般我们在计算的“角度”， 180 度角约等于 3.14 径度！嗯！那我们就来写一下这个程序吧！

```
[root@study ~]# vim sin.c
#include &lt;stdio.h&gt;
#include &lt;math.h&gt;
int main（void）
{
        float value;
        value = sin （ 3.14 / 2 ）;
        printf（"%f\n",value）;
}
```

上面这个文件的内容可以在下面取得！

- http://linux.vbird.org/linux_basic/0520source/sin.c

那要如何编译这支程序呢？我们先直接编译看看：

```
[root@study ~]# gcc sin.c
# 新的 GCC 会主动将函数抓进来给你用，所以只要加上 include &lt;math.h&gt; 就好了！
```

新版的 GCC 会主动帮你将所需要的函数库抓进来编译，所以不会出现怪异的错误讯息！ 事实上，数学函数库使用的是 libm.so 这个函数库，你最好在编译的时候将这个函数库纳进去比较好～另外要注意， 这个函数库放置的地方是系统默认会去找的 /lib, /lib64 ，所以你无须使用下面的 -L 去加入搜寻的目录！ 而 libm.so 在编译的写法上，使用的是 -lm （lib 简写为 l 喔！） 喔！因此就变成：

- 编译时加入额外函数库链接的方式：

```
[root@study ~]# gcc sin.c -lm -L/lib -L/lib64  &lt;==重点在 -lm
[root@study ~]# ./a.out                         &lt;==尝试执行新文件！
1.000000
```

特别注意，使用 gcc 编译时所加入的那个 -lm 是有意义的，他可以拆开成两部份来看：

- -l ：是“加入某个函数库（library）”的意思，
- m ：则是 libm.so 这个函数库，其中， lib 与扩展名（.a 或 .so）不需要写

所以 -lm 表示使用 libm.so （或 libm.a） 这个函数库的意思～至于那个 -L 后面接的路径呢？这表示：“我要的函数库 libm.so 请到 /lib 或 /lib64 里面搜寻！”

上面的说明很清楚了吧！不过，要注意的是，由于 Linux 默认是将函数库放置在 /lib 与 /lib64 当中，所以你没有写 -L/lib 与 -L/lib64 也没有关系的！不过，万一哪天你使用的函数库并非放置在这两个目录下，那么 -L/path 就很重要了！否则会找不到函数库喔！

除了链接的函数库之外，你或许已经发现一个奇怪的地方，那就是在我们的 sin.c 当中第一行“ #include <stdio.h>”，这行说的是要将一些定义数据由 stdio.h 这个文件读入，这包括 printf 的相关设置。这个文件其实是放置在 /usr/include/stdio.h 的！那么万一这个文件并非放置在这里呢？那么我们就可以使用下面的方式来定义出要读取的 include 文件放置的目录：

```
[root@study ~]# gcc sin.c -lm -I/usr/include
```

-I/path 后面接的路径（ Path ）就是设置要去搜寻相关的 include 文件的目录啦！不过，同样的，默认值是放置在 /usr/include 下面，除非你的 include 文件放置在其他路径，否则也可以略过这个项目！

通过上面的几个小范例，你应该对于 gcc 以及源代码有一定程度的认识了，再接下来，我们来稍微整理一下 gcc 的简易使用方法吧！

21.2.4 gcc 的简易用法 （编译、参数与链结）

前面说过， gcc 为 Linux 上面最标准的编译器，这个 gcc 是由 GNU 计划所维护的，有兴趣的朋友请自行前往参考。既然 gcc 对于 Linux 上的 Open source 是这么样的重要，所以下面我们就列举几个 gcc 常见的参数，如此一来大家应该更容易了解源代码的各项功能吧！

```
# 仅将源代码编译成为目标文件，并不制作链接等功能：
[root@study ~]# gcc -c hello.c
# 会自动的产生 hello.o 这个文件，但是并不会产生 binary 可执行文件。

# 在编译的时候，依据作业环境给予最优化执行速度
[root@study ~]# gcc -O hello.c -c
# 会自动的产生 hello.o 这个文件，并且进行最优化喔！

# 在进行 binary file 制作时，将链接的函数库与相关的路径填入
[root@study ~]# gcc sin.c -lm -L/lib -I/usr/include
# 这个指令较常下达在最终链接成 binary file 的时候，
# -lm 指的是 libm.so 或 libm.a 这个函数库文件；
# -L 后面接的路径是刚刚上面那个函数库的搜寻目录；
# -I 后面接的是源代码内的 include 文件之所在目录。

# 将编译的结果输出成某个特定文件名
[root@study ~]# gcc -o hello hello.c
# -o 后面接的是要输出的 binary file 文件名

# 在编译的时候，输出较多的讯息说明
[root@study ~]# gcc -o hello hello.c -Wall
# 加入 -Wall 之后，程序的编译会变的较为严谨一点，所以警告讯息也会显示出来！
```

比较重要的大概就是这一些。另外，我们通常称 -Wall 或者 -O 这些非必要的参数为旗标（FLAGS），因为我们使用的是 C 程序语言，所以有时候也会简称这些旗标为 CFLAGS ，这些变量偶尔会被使用的喔！尤其是在后头会介绍的 make 相关的用法时，更是重要的很呐！ ^_^

21.3 用 make 进行宏编译

在本章一开始我们提到过 make 的功能是可以简化编译过程里面所下达的指令，同时还具有很多很方便的功能！那么下面咱们就来试看看使用 make 简化下达编译指令的流程吧！

21.3.1 为什么要用 make

先来想像一个案例，假设我的可执行文件里面包含了四个源代码文件，分别是 main.c haha.c sin_value.c cos_value.c 这四个文件，这四个文件的目的是：

- main.c ：主要的目的是让使用者输入角度数据与调用其他三支副程序；
- haha.c ：输出一堆有的没有的讯息而已；
- sin_value.c ：计算使用者输入的角度（360） sin 数值；
- cos_value.c ：计算使用者输入的角度（360） cos 数值。

这四个文件你可以到 http://linux.vbird.org/linux_basic/0520source/main.tgz 来下载。由于这四个文件里面包含了相关性，并且还用到数学函数在里面，所以如果你想要让这个程序可以跑， 那么就需要这样编译：

```
# 1\. 先进行目标文件的编译，最终会有四个 *.o 的文件名出现：
[root@study ~]# gcc -c main.c
[root@study ~]# gcc -c haha.c
[root@study ~]# gcc -c sin_value.c
[root@study ~]# gcc -c cos_value.c

# 2\. 再进行链接成为可执行文件，并加入 libm 的数学函数，以产生 main 可执行文件：
[root@study ~]# gcc -o main main.o haha.o sin_value.o cos_value.o -lm

# 3\. 本程序的执行结果，必须输入姓名、360 度角的角度值来计算：
[root@study ~]# ./main
Please input your name: VBird  &lt;==这里先输入名字
Please enter the degree angle （ex&gt; 90）: 30   &lt;==输入以 360 度角为主的角度
Hi, Dear VBird, nice to meet you.    &lt;==这三行为输出的结果喔！
The Sin is:  0.50
The Cos is:  0.87
```

编译的过程需要进行好多动作啊！而且如果要重新编译，则上述的流程得要重新来一遍，光是找出这些指令就够烦人的了！ 如果可以的话，能不能一个步骤就给他完成上面所有的动作呢？那就利用 make 这个工具吧！ 先试看看在这个目录下创建一个名为 makefile 的文件，内容如下：

```
# 1\. 先编辑 makefile 这个规则档，内容只要作出 main 这个可执行文件
[root@study ~]# vim makefile
main: main.o haha.o sin_value.o cos_value.o
    gcc -o main main.o haha.o sin_value.o cos_value.o -lm
# 注意：第二行的 gcc 之前是 &lt;tab&gt; 按键产生的空格喔！

# 2\. 尝试使用 makefile 制订的规则进行编译的行为：
[root@study ~]# rm -f main *.o   &lt;==先将之前的目标文件去除
[root@study ~]# make
cc    -c -o main.o main.c
cc    -c -o haha.o haha.c
cc    -c -o sin_value.o sin_value.c
cc    -c -o cos_value.o cos_value.c
gcc -o main main.o haha.o sin_value.o cos_value.o -lm
# 此时 make 会去读取 makefile 的内容，并根据内容直接去给他编译相关的文件啰！

# 3\. 在不删除任何文件的情况下，重新执行一次编译的动作：
[root@study ~]# make
make: `main' is up to date.
# 看到了吧！是否很方便呢！只会进行更新 （update） 的动作而已。
```

或许你会说：“如果我创建一个 shell script 来将上面的所有动作都集结在一起，不是具有同样的效果吗？”呵呵！ 效果当然不一样，以上面的测试为例，我们仅写出 main 需要的目标文件，结果 make 会主动的去判断每个目标文件相关的源代码文件，并直接予以编译，最后再直接进行链接的动作！ 真的是很方便啊！此外，如果我们更动过某些源代码文件，则 make 也可以主动的判断哪一个源代码与相关的目标文件文件有更新过， 并仅更新该文件，如此一来，将可大大的节省很多编译的时间呢！要知道，某些程序在进行编译的行为时，会消耗很多的 CPU 资源呢！所以说， make 有这些好处：

- 简化编译时所需要下达的指令；
- 若在编译完成之后，修改了某个源代码文件，则 make 仅会针对被修改了的文件进行编译，其他的 object file 不会被更动；
- 最后可以依照相依性来更新 （update） 可执行文件。

既然 make 有这么多的优点，那么我们当然就得好好的了解一下 make 这个令人关心的家伙啦！而 make 里面最需要注意的大概就是那个规则文件，也就是 makefile 这个文件的语法啦！所以下面我们就针对 makefile 的语法来加以介绍啰。

21.3.2 makefile 的基本语法与变量

make 的语法可是相当的多而复杂的，有兴趣的话可以到 GNU [1] 去查阅相关的说明，鸟哥这里仅列出一些基本的规则，重点在于让读者们未来在接触源代码时，不会太紧张啊！ 好了，基本的 makefile 规则是这样的：

```
标的（target）: 目标文件1 目标文件2
&lt;tab&gt;   gcc -o 欲创建的可执行文件 目标文件1 目标文件2
```

那个标的 （target） 就是我们想要创建的信息，而目标文件就是具有相关性的 object files ，那创建可执行文件的语法就是以 <tab> 按键开头的那一行！特别给他留意喔，“命令列必须要以 tab 按键作为开头”才行！他的规则基本上是这样的：

- 在 makefile 当中的 # 代表注解；
- <tab> 需要在命令行（例如 gcc 这个编译器指令）的第一个字符；
- 标的（target）与相依文件（就是目标文件）之间需以“:”隔开。

同样的，我们以刚刚上一个小节的范例进一步说明，如果我想要有两个以上的执行动作时，例如下达一个指令就直接清除掉所有的目标文件与可执行文件，该如何制作呢？

```
# 1\. 先编辑 makefile 来创建新的规则，此规则的标的名称为 clean ：
[root@study ~]# vi makefile
main: main.o haha.o sin_value.o cos_value.o
    gcc -o main main.o haha.o sin_value.o cos_value.o -lm
clean:
    rm -f main main.o haha.o sin_value.o cos_value.o

# 2\. 以新的标的 （clean） 测试看看执行 make 的结果：
[root@study ~]# make clean  &lt;==就是这里！通过 make 以 clean 为标的
rm -rf main main.o haha.o sin_value.o cos_value.o
```

如此一来，我们的 makefile 里面就具有至少两个标的，分别是 main 与 clean ，如果我们想要创建 main 的话，输入“make main”，如果想要清除有的没的，输入“make clean”即可啊！而如果想要先清除目标文件再编译 main 这个程序的话，就可以这样输入：“make clean main”，如下所示：

```
[root@study ~]# make clean main
rm -rf main main.o haha.o sin_value.o cos_value.o
cc    -c -o main.o main.c
cc    -c -o haha.o haha.c
cc    -c -o sin_value.o sin_value.c
cc    -c -o cos_value.o cos_value.c
gcc -o main main.o haha.o sin_value.o cos_value.o -lm
```

这样就很清楚了吧！但是，你是否会觉得，咦！ makefile 里面怎么重复的数据这么多啊！没错！所以我们可以再借由 shell script 那时学到的“变量”来更简化 makefile 喔：

```
[root@study ~]# vi makefile
LIBS = -lm
OBJS = main.o haha.o sin_value.o cos_value.o
main: ${OBJS}
        gcc -o main ${OBJS} ${LIBS}
clean:
        rm -f main ${OBJS}
```

与 bash shell script 的语法有点不太相同，变量的基本语法为：

1. 变量与变量内容以“=”隔开，同时两边可以具有空格；
2. 变量左边不可以有 <tab> ，例如上面范例的第一行 LIBS 左边不可以是 <tab>；
3. 变量与变量内容在“=”两边不能具有“:”；
4. 在习惯上，变量最好是以“大写字母”为主；
5. 运用变量时，以 ${变量} 或 $（变量） 使用；
6. 在该 shell 的环境变量是可以被套用的，例如提到的 CFLAGS 这个变量！
7. 在命令行界面也可以给予变量。

由于 gcc 在进行编译的行为时，会主动的去读取 CFLAGS 这个环境变量，所以，你可以直接在 shell 定义出这个环境变量，也可以在 makefile 文件里面去定义，更可以在命令行当中给予这个咚咚呢！例如：

```
[root@study ~]# CFLAGS="-Wall" make clean main
# 这个动作在上 make 进行编译时，会去取用 CFLAGS 的变量内容！
```

也可以这样：

```
[root@study ~]# vi makefile
LIBS = -lm
OBJS = main.o haha.o sin_value.o cos_value.o
CFLAGS = -Wall
main: ${OBJS}
    gcc -o main ${OBJS} ${LIBS}
clean:
    rm -f main ${OBJS}
```

咦！我可以利用命令行进行环境变量的输入，也可以在文件内直接指定环境变量，那万一这个 CFLAGS 的内容在命令行与 makefile 里面并不相同时，以那个方式输入的为主？呵呵！问了个好问题啊！ 环境变量取用的规则是这样的：

1. make 命令行后面加上的环境变量为优先；
2. makefile 里面指定的环境变量第二；
3. shell 原本具有的环境变量第三。

此外，还有一些特殊的变量需要了解的喔：

- $@：代表目前的标的（target）

所以我也可以将 makefile 改成：

```
[root@study ~]# vi makefile
LIBS = -lm
OBJS = main.o haha.o sin_value.o cos_value.o
CFLAGS = -Wall
main: ${OBJS}
    gcc -o $@ ${OBJS} ${LIBS}   &lt;==那个 $@ 就是 main ！
clean:
    rm -f main ${OBJS}
```

这样是否稍微了解了 makefile （也可能是 Makefile） 的基本语法？这对于你未来自行修改源代码的编译规则时，是很有帮助的喔！^_^！

21.4 Tarball 的管理与建议

在我们知道了源代码的相关信息之后，再来要了解的自然就是如何使用具有源代码的 Tarball 来创建一个属于自己的软件啰！从前面几个小节的说明当中，我们晓得其实 Tarball 的安装是可以跨平台的，因为 C 语言的程序码在各个平台上面是可以共通的， 只是需要的编译器可能并不相同而已。例如 Linux 上面用 gcc 而 Windows 上面也有相关的 C 编译器啊～所以呢，同样的一组源代码，既可以在 CentOS Linux 上面编译，也可以在 SuSE Linux 上面编译，当然，也可以在大部分的 Unix 平台上面编译成功的！

如果万一没有编译成功怎么办？很简单啊，通过修改小部分的程序码 （通常是因为很小部分的异动而已） 就可以进行跨平台的移植了！也就是说，刚刚我们在 Linux 下面写的程序“理论上，是可以在 Windows 上面编译的！”这就是源代码的好处啦！所以说，如果朋友们想要学习程序语言的话， 鸟哥个人是比较建议学习“具有跨平台能力的程序语言”，例如 C 就是很不错的一个！

唉啊！又扯远了～赶紧拉回来继续说明我们的 Tarball 啦！

21.4.1 使用源代码管理软件所需要的基础软件

从源代码的说明我们晓得要制作一个 binary program 需要很多咚咚的呢！这包括下面这些基础的软件：

- gcc 或 cc 等 C 语言编译器 （compiler）：

没有编译器怎么进行编译的动作？所以 C compiler 是一定要有的。不过 Linux 上面有众多的编译器，其中当然以 GNU 的 gcc 是首选的自由软件编译器啰！事实上很多在 Linux 平台上面发展的软件的源代码，原本就是以 gcc 为底来设计的呢。

- make 及 autoconfig 等软件：

一般来说，以 Tarball 方式释出的软件当中，为了简化编译的流程，通常都是配合前几个小节提到的 make 这个指令来依据目标文件的相依性而进行编译。但是我们也知道说 make 需要 makefile 这个文件的规则，那由于不同的系统里面可能具有的基础软件环境并不相同， 所以就需要侦测使用者的作业环境，好自行创建一个 makefile 文件。这个自行侦测的小程序也必须要借由 autoconfig 这个相关的软件来辅助才行。

- 需要 Kernel 提供的 Library 以及相关的 Include 文件：

从前面的源代码编译过程，我们晓得函数库 （library） 的重要性，同时也晓得有 include 文件的存在。很多的软件在发展的时候都是直接取用系统核心提供的函数库与 include 文件的，这样才可以与这个操作系统相容啊！尤其是在“驱动程序方面的模块 ”，例如网卡、声卡、

USB 等驱动程序在安装的时候，常常是需要核心提供的相关信息的。在 Red Hat 的系统当中（包含 Fedora/CentOS 等系列），这个核心相关的功能通常都是被包含在 kernel-source 或 kernel-header 这些软件名称当中，所以记得要安装这些软件喔！

虽然 Tarball 的安装上面相当的简单，如同我们前面几个小节的例子，只要顺着开发商提供的 README 与 INSTALL 文件所载明的步骤来进行，安装是很容易的。但是我们却还是常常会在 BBS 或者是新闻群组当中发现这些留言："我在执行某个程序的侦测文件时，他都会告诉我没有 gcc 这个软件，这是怎么回事？"还有："我没有办法使用 make 耶！这是什么问题？"呵呵！ 这就是没有安装上面提到的那些基础软件啦！

咦！为什么使用者不安装这些软件啊？这是因为目前的 Linux distribution 大多已经偏向于桌面电脑的使用（非服务器端），他们希望使用者能够按照厂商自己的希望来安装相关的软件即可， 所以通常"默认"是没有安装 gcc 或者是 make 等软件的。所以啦，如果你希望未来可以自行安装一些以 Tarball 方式释出的软件时，记得请自行挑选想要安装的软件名称喔！例如在 CentOS 或者是 Red Hat 当中记得选择 Development Tools 以及 Kernel Source Development 等相关字眼的软件群集呢。

那万一我已经安装好一部 Linux 主机，但是使用的是默认值所安装的软件，所以没有 make, gcc 等咚咚，该如何是好？呵呵！问题其实不大啦，目前使用最广泛的 CentOS/Fedora 或者是 Red Hat 大多是以 RPM（下一章会介绍）来安装软件的，所以，你只要拿出当初安装 Linux 时的原版光盘，然后以下一章介绍的 RPM 来一个一个的加入到你的 Linux 主机里面就好啦！很简单的啦！ 尤其现在又有 yum 这玩意儿，更方便吶！

在 CentOS 当中，如果你已经有网络可以连上 Internet 的话，那么就可以使用下一章会谈到的 yum 啰！ 通过 yum 的软件群组安装功能，你可以这样做：

- 如果是要安装 gcc 等软件发展工具，请使用" yum groupinstall "Development Tools" "
- 若待安装的软件需要图形接口支持，一般还需要" yum groupinstall "X Software Development" "
- 若安装的软件较旧，可能需要" yum groupinstall "Legacy Software Development" "

大概就是这样，更多的信息请参考下一章的介绍喔。

21.4.2 Tarball 安装的基本步骤

我们提过以 Tarball 方式释出的软件是需要重新编译可执行的 binary program 的。而 Tarball 是以 tar 这个指令来打包与压缩的文件，所以啦，当然就需要先将 Tarball 解压缩，然后到源代码所在的目录下进行 makefile 的创建，再以 make 来进行编译与安装的动作啊！所以整个安装的基础动作大多是这样的：

1. 取得原始文件：将 tarball 文件在 /usr/local/src 目录下解压缩；
2. 取得步骤流程：进入新创建的目录下面，去查阅 INSTALL 与 README 等相关文件内容（很重要的步骤！）；
3. 相依属性软件安装：根据 INSTALL/README 的内容察看并安装好一些相依的软件（非

必要）；
4. 创建 makefile：以自动侦测程序 （configure 或 config） 侦测作业环境，并创建 Makefile 这个文件；
5. 编译：以 make 这个程序并使用该目录下的 Makefile 做为他的参数配置文件，来进行 make （编译或其他） 的动作；
6. 安装：以 make 这个程序，并以 Makefile 这个参数配置文件，依据 install 这个标的 （target） 的指定来安装到正确的路径！

注意到上面的第二个步骤，通常在每个软件在释出的时候，都会附上 INSTALL 或者是 README 这种文件名的说明文档，这些说明文档请“确实详细的” 阅读过一遍，通常这些文件会记录这个软件的安装要求、软件的工作项目、 与软件的安装参数设置及技巧等，只要仔细的读完这些文件，基本上，要安装好 tarball 的文件，都不会有什么大问题啰。

至于 makefile 在制作出来之后，里头会有相当多的标的 （target），最常见的就是 install 与 clean 啰！通常“make clean”代表着将目标文件 （object file） 清除掉，“make”则是将源代码进行编译而已。 注意喔！编译完成的可可执行文件与相关的配置文件还在源代码所在的目录当中喔！因此，最后要进行“make install”来将编译完成的所有咚咚都给他安装到正确的路径去，这样就可以使用该软件啦！

OK！我们下面约略提一下大部分的 tarball 软件之安装的指令下达方式：

1. ./configure 这个步骤就是在创建 Makefile 这个文件啰！通常程序开发者会写一支 scripts 来检查你的 Linux 系统、相关的软件属性等等，这个步骤相当的重要， 因为未来你的安装信息都是这一步骤内完成的！另外，这个步骤的相关信息应该要参考一下该目录下的 README 或 INSTALL 相关的文件！

2. make clean make 会读取 Makefile 中关于 clean 的工作。这个步骤不一定会有，但是希望执行一下，因为他可以去除目标文件！因为谁也不确定源代码里面到底有没有包含上次编译过的目标文件 （*.o） 存在，所以当然还是清除一下比较妥当的。 至少等一下新编译出来的可执行文件我们可以确定是使用自己的机器所编译完成的嘛！

3. make make 会依据 Makefile 当中的默认工作进行编译的行为！编译的工作主要是进行 gcc 来将源代码编译成为可以被执行的 object files ，但是这些 object files 通常还需要一些函数库之类的 link 后，才能产生一个完整的可执行文件！使用 make 就是要将源代码编译成为可以被执行的可可执行文件，而这个可可执行文件会放置在目前所在的目录之下， 尚未被安装到预定安装的目录中；

4. make install 通常这就是最后的安装步骤了，make 会依据 Makefile 这个文件里面关于 install 的项目，将上一个步骤所编译完成的数据给他安装到预定的目录中，就完成安装啦！

请注意，上面的步骤是一步一步来进行的，而其中只要一个步骤无法成功，那么后续的步骤就完全没有办法进行的！ 因此，要确定每一的步骤都是成功的才可以！举个例子来说，万一今天你在 ./configure 就不成功了，那么就表示 Makefile 无法被创建起来，要知道，后面的步

骤都是根据 Makefile 来进行的，既然无法创建 Makefile，后续的步骤当然无法成功啰！

另外，如果在 make 无法成功的话，那就表示原始文件无法被编译成可可执行文件，那么 make install 主要是将编译完成的文件给他放置到文件系统中的，既然都没有可用的可执行文件了，怎么进行安装？ 所以啰，要每一个步骤都正确无误才能往下继续做！此外，如果安装成功， 并且是安装在独立的一个目录中，例如 /usr/local/packages 这个目录中好了，那么你就必需手动的将这个软件的 man page 给他写入 /etc/man_db.conf 里面去。

21.4.3 一般 Tarball 软件安装的建议事项 （如何移除？升级？）

或许你已经发现了也说不定，那就是为什么前一个小节里面， Tarball 要在 /usr/local/src 里面解压缩呢？基本上，在默认的情况下，原本的 Linux distribution 释出安装的软件大多是在 /usr 里面的，而使用者自行安装的软件则建议放置在 /usr/local 里面。这是考虑到管理使用者所安装软件的便利性。

怎么说呢？我们晓得几乎每个软件都会提供线上说明的服务，那就是 info 与 man 的功能。在默认的情况下， man 会去搜寻 /usr/local/man 里面的说明文档， 因此，如果我们将软件安装在 /usr/local 下面的话，那么自然安装完成之后， 该软件的说明文档就可以被找到了。此外，如果你所管理的主机其实是由多人共同管理的， 或者是如同学校里面，一部主机是由学生管理的，但是学生总会毕业吧？ 所以需要进行交接，如果大家都将软件安装在 /usr/local 下面，那么管理上不就显的特别的容易吗！

所以啰，通常我们会建议大家将自己安装的软件放置在 /usr/local 下，至于源代码 （Tarball） 则建议放置在 /usr/local/src （src 为 source 的缩写）下面啊。

再来，让我们先来看一看 Linux distribution 默认的安装软件的路径会用到哪些？我们以 apache 这个软件来说明的话 （apache 是 WWW 服务器软件，详细的数据请参考服务器架设篇。你的系统不见得有装这个软件）：

- /etc/httpd
- /usr/lib
- /usr/bin
- /usr/share/man

我们会发现软件的内容大致上是摆在 etc, lib, bin, man 等目录当中，分别代表“配置文件、函数库、可执行文件、线上说明文档”。 好了，那么你是以 tarball 来安装时呢？如果是放在默认的 /usr/local 里面，由于 /usr/local 原本就默认这几个目录了，所以你的数据就会被放在：

- /usr/local/etc
- /usr/local/bin
- /usr/local/lib
- /usr/local/man

但是如果你每个软件都选择在这个默认的路径下安装的话， 那么所有的软件的文件都将放置在这四个目录当中，因此，如果你都安装在这个目录下的话， 那么未来再想要升级或移除的时候，就会比较难以追查文件的来源啰！ 而如果你在安装的时候选择的是单独的目录，例如我将 apache 安装在 /usr/local/apache 当中，那么你的文件目录就会变成：

- /usr/local/apache/etc
- /usr/local/apache/bin
- /usr/local/apache/lib
- /usr/local/apache/man

呵呵！单一软件的文件都在同一个目录之下，那么要移除该软件就简单的多了！ 只要将该目录移除即可视为该软件已经被移除啰！以上面为例，我想要移除 apache 只要下达“rm -rf /usr/local/apache” 就算移除这个软件啦！当然啰，实际安装的时候还是得视该软件的 Makefile 里头的 install 信息才能知道到底他的安装情况为何的。因为例如 sendmail 的安装就很麻烦......

这个方式虽然有利于软件的移除，但不晓得你有没有发现，我们在执行某些指令的时候，与该指令是否在 PATH 这个环境变量所记录的路径有关，以上面为例，我的 /usr/local/apache/bin 肯定是不在 PATH 里面的，所以执行 apache 的指令就得要利用绝对路径了，否则就得将这个 /usr/local/apache/bin 加入 PATH 里面。另外，那个 /usr/local/apache/man 也需要加入 man page 搜寻的路径当中啊！

除此之外， Tarball 在升级的时候也是挺困扰的，怎么说呢？我们还是以 apache 来说明好了。WWW 服务器为了考虑互动性，所以通常会将 PHP+MySQL+Apache 一起安装起来 （详细的信息请参考服务器架设篇） ，果真如此的话，那么每个软件在安装的时候“都有一定的顺序与程序！” 因为他们三者之间具有相关性，所以安装时必需要三者同时考虑到他们的函数库与相关的编译参数。

假设今天我只要升级 PHP 呢？有的时候因为只有涉及动态函数库的升级，那么我只要升级 PHP 即可！其他的部分或许影响不大。但是如果今天 PHP 需要重新编译的模块比较多，那么可能会连带的，连 Apache 这个程序也需要重新编译过才行！真是有点给他头痛的！没办法啦！使用 tarball 确实有他的优点啦，但是在这方面，确实也有他一定的伤脑筋程度。

由于 Tarball 在升级与安装上面具有这些特色，亦即 Tarball 在反安装上面具有比较高的难度（如果你没有好好规划的话～），所以，为了方便 Tarball 的管理，通常鸟哥会这样建议使用者：

1. 最好将 tarball 的原始数据解压缩到 /usr/local/src 当中；
2. 安装时，最好安装到 /usr/local 这个默认路径下；
3. 考虑未来的反安装步骤，最好可以将每个软件单独的安装在 /usr/local 下面；

4. 为安装到单独目录的软件之 man page 加入 man path 搜寻： 如果你安装的软件放置到 /usr/local/software/ ，那么 man page 搜寻的设置中，可能就得要在 /etc/man_db.conf 内的 40~50 行左右处，写入如下的一行：

   > MANPATH_MAP /usr/local/software/bin /usr/local/software/man

   这样才可以使用 man 来查询该软件的线上文件啰！



Tips 时至今日，老实说，真的不太需要有 tarball 的安装了！CentOS/Fedora 有个 RPM 补遗计划，就是俗称的 EPEL 计划，相关网址说明如下： https://fedoraproject.org/wiki/EPEL～一般学界会用到的软件都在里头～ 除非你要用的软件是专属软件 （要钱的） 或者是比较冷门的软件，否则都有好心的网友帮我们打包好了啦！ ^_^

21.4.4 一个简单的范例、利用 ntp 来示范

读万卷书不如行万里路啊！所以当然我们就来给他测试看看，看你是否真的了解了如何利用 Tarball 来安装软件呢？我们利用时间服务器 （network time protocol） ntp 这个软件来测试安装看看。先请到 http://www.ntp.org/downloads.html 这个目录去下载文件，请下载最新版本的文件即可。或者直接到鸟哥的网站下载 2015/06 公告释出的稳定版本：

> http://linux.vbird.org/linux_basic/0520source/ntp-4.2.8p3.tar.gz

假设我对这个软件的要求是这样的：

- 假设 ntp-4...tar.gz 这个文件放置在 /root 这个目录下；
- 源代码请解开在 /usr/local/src 下面；
- 我要安装到 /usr/local/ntp 这个目录中；

那么你可以依照下面的步骤来安装测试看看 （如果可以的话，请你不要参考下面的文件数据， 先自行安装过一遍这个软件，然后再来对照一下鸟哥的步骤喔！）。

- 解压缩下载的 tarball ，并参阅 README/INSTALL 文件

```
[root@study ~]# cd /usr/local/src   &lt;==切换目录
[root@study src]# tar -zxvf /root/ntp-4.2.8p3.tar.gz  &lt;==解压缩到此目录
ntp-4.2.8p3/         &lt;==会创建这个目录喔！
ntp-4.2.8p3/CommitLog
....（下面省略）....
[root@study src]# cd ntp-4.2.8p3
[root@study ntp-4.2.8p3]# vi INSTALL  &lt;==记得 README 也要看一下！
# 特别看一下 28 行到 54 行之间的安装简介！可以了解如何安装的流程喔！
```

- 检查 configure 支持参数，并实际创建 makefile 规则档

```
[root@study ntp*]# ./configure --help &#124; more  &lt;==查询可用的参数有哪些
  --prefix=PREFIX         install architecture-independent files in PREFIX
  --enable-all-clocks     + include all suitable non-PARSE clocks:
  --enable-parse-clocks   - include all suitable PARSE clocks:
# 上面列出的是比较重要的，或者是你可能需要的参数功能！

[root@study ntp*]# ./configure --prefix=/usr/local/ntp \
&gt;  --enable-all-clocks --enable-parse-clocks  &lt;==开始创建makefile
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
....（中间省略）....
checking for gcc... gcc           &lt;==也有找到 gcc 编译器了！
....（中间省略）....
config.status: creating Makefile  &lt;==现在知道这个重要性了吧？
config.status: creating config.h
config.status: creating evconfig-private.h
config.status: executing depfiles commands
config.status: executing libtool commands
```

一般来说 configure 设置参数较重要的就是那个 --prefix=/path 了，--prefix 后面接的路径就是“这个软件未来要安装到那个目录去？”如果你没有指定 --prefix=/path 这个参数，通常默认参数就是 /usr/local 至于其他的参数意义就得要参考 ./configure --help 了！ 这个动作完成之后会产生 makefile 或 Makefile 这个文件。当然啦，这个侦测检查的过程会显示在屏幕上， 特别留意关于 gcc 的检查，还有最重要的是最后需要成功的创建起 Makefile 才行！

- 最后开始编译与安装噜！

```
[root@study ntp*]# make clean; make
[root@study ntp*]# make check
[root@study ntp*]# make install
# 将数据给他安装在 /usr/local/ntp 下面
```

整个动作就这么简单，你完成了吗？完成之后到 /usr/local/ntp 你发现了什么？

21.4.5 利用 patch 更新源代码

我们在本章一开始介绍了为何需要进行软件的升级，这是很重要的喔！那假如我是以 Tarball 来进行某个软件的安装，那么是否当我要升级这个软件时，就得要下载这个软件的完整全新的 Tarball 呢？举个例子来说，鸟哥的讨论区 http://phorum.vbird.org 这个网址，这个讨论区是以 phpBB 这个软件来架设的，而鸟哥的讨论区版本为 3.1.4 ，目前 （2015/09） 最新释出的版本则是 phpbb 3.1.5 。那我是否需要下载全新的 phpbb3.1.5.tar.gz 这个文件来更新原本的旧程序呢？

事实上，当我们发现一些软件的漏洞，通常是某一段程序码写的不好所致。因此， 所谓的“更新源代码”常常是只有更改部分文件的小部分内容而已。既然如此的话， 那么我们是否可以就那些被更动的文件来进行修改就可以咯？也就是说， 旧版本到新版本间没有更动过的文件就不要理他，仅将有修订过的文件部分来处理即可。

这有什么好处呢？首先，没有更动过的文件的目标文件 （object file） 根本就不需要重新编译，而且有更动过的文件又可以利用 make 来自动 update （更新） ，如此一来，我们原先的设置 （makefile 文件里面的规则） 将不需要重新改写或侦测！可以节省很多宝贵的时间呢（例如后续章节会提到的核心的编译！）

从上面的说明当中，我们可以发现，如果可以将旧版的源代码数据改写成新版的版本， 那么就能直接编译了，而不需要将全部的新版 Tarball 重新下载一次呢！可以节省带宽与时间说！那么如何改写源代码？ 难道要我们一个文件一个文件去参考然后修订吗？当然没有这么没人性！

我们在第十一章、正则表达式的时候有提到一个比对文件的指令，那就是 diff，这个指令可以将“两个文件之间的差异性列出来”呢！那我们也知道新旧版本的文件之间， 其实只有修改一些程序码而已，那么我们可以通过 diff 比对出新旧版本之间的文字差异，然后再以相关的指令来将旧版的文件更新吗？ 呵呵！当然可以啦！那就是 patch 这个指令啦！很多的软件开发商在更新了源代码之后，几乎都会释出所谓的 patch file，也就是直接将源代码 update 而已的一个方式喔！我们下面以一个简单的范例来说明给你了解喔！

关于 diff 与 patch 的基本用法我们在第十一章都谈过了，所以这里不再就这两个指令的语法进行介绍， 请回去参阅该章的内容。这里我们来举个案例解释一下好了。假设我们刚刚计算三角函数的程序 （main） 历经多次改版， 0.1 版仅会简单的输出， 0.2 版的输出就会含有角度值，因此这两个版本的内容不相同。如下所示，两个文件的意义为：

- http://linux.vbird.org/linux_basic/0520source/main-0.1.tgz ：main 的 0.1 版；
- http://linux.vbird.org/linux_basic/0520source/main_0.1_to_0.2.patch ：main 由 0.1 升级到 0.2 的 patch file；

请您先下载这两个文件，并且解压缩到你的 /root 下面。你会发现系统产生一个名为 main-0.1 的目录。 该目录内含有五个文件，就是刚刚的程序加上一个 Makefile 的规则文件。你可以到该目录下去看看 Makefile 的内容， 在这一版当中含有 main 与 clean 两个标的功能而已。至于 0.2 版则加入了 install 与 uninstall 的规则设置。 接下来，请看一下我们的作法啰：

- 测试旧版程序的功能

```
[root@study ~]# tar -zxvf main-0.1.tgz
[root@study ~]# cd main-0.1
[root@study main-0.1]# make clean main
[root@study main-0.1]# ./main
version 0.1
Please input your name: VBird
Please enter the degree angle （ex&gt; 90） : 45
Hi, Dear VBird, nice to meet you.
The Sin is:  0.71
The Cos is:  0.71
```

与之前的结果非常类似，只是鸟哥将 Makefile 直接给您了！但如果你下达 make install 时，系统会告知没有 install 的 target 啊！而且版本是 0.1 也告知了。那么如何更新到 0.2 版呢？通过这个 patch 文件吧！这个文件的内容有点像这样：

- 查阅 patch file 内容

```
[root@study main-0.1]# vim ~/main_0.1_to_0.2.patch
diff -Naur main-0.1/cos_value.c main-0.2/cos_value.c
--- main-0.1/cos_value.c        2015-09-04 14:46:59.200444001 +0800
+++ main-0.2/cos_value.c        2015-09-04 14:47:10.215444000 +0800
@@ -7,5 +7,5 @@
 {
        float value;
....（下面省略）....
```

上面表格内有个底线的部分，那代表使用 diff 去比较时，被比较的两个文件所在路径，这个路径非常的重要喔！ 因为 patch 的基本语法如下：

> patch -p数字 < patch_file

特别留意那个“ -p数字”，那是与 patch_file 里面列出的文件名有关的信息。假如在 patch_file 第一行写的是这样：

> * /home/guest/example/expatch.old

那么当我下达“ patch -p0 < patch_file ”时，则更新的文件是“ /home/guest/example/expatch.old ”，如果“ patch -p1 < patch_file”，则更新的文件为“home/guest/example/expatch.old”，如果“patch -p4 < patch_file”则更新“expatch.old”，也就是说， -pxx 那个 xx 代表“拿掉几个斜线（/）”的意思！这样可以理解了吗？ 好了，根据刚刚上头的数据，我们可以发现比较的文件是在 main-0.1/xxx 与 main-0.2/xxx ， 所以说，如果你是在 main-0.1 下面，并且想要处理更新时，就得要拿掉一个目录 （因为并没有 main-0.2 的目录存在， 我们是在当前的目录进行更新的！），因此使用的是 -p1 才对喔！所以：

- 更新源代码，并且重新编译程序！

```
[root@study main-0.1]# patch -p1 &lt; ../main_0.1_to_0.2.patch
patching file cos_value.c
patching file main.c
patching file Makefile
patching file sin_value.c
# 请注意，鸟哥目前所在目录是在 main-0.1 下面喔！注意与 patch 文件的相对路径！
# 虽然有五个文件，但其实只有四个文件有修改过喔！上面显示有改过的文件！

[root@study main-0.1]# make clean main
[root@study main-0.1]# ./main
version 0.2
Please input your name: VBird
Please enter the degree angle （ex&gt; 90）: 45
Hi, Dear VBird, nice to meet you.
The sin（45.000000） is:  0.71
The cos（45.000000） is:  0.71
# 你可以发现，输出的结果中版本变了，输出信息多了括号 （） 喔！

[root@study main-0.1]# make install   &lt;==将他安装到 /usr/local/bin 给大家用
cp -a main /usr/local/bin
[root@study main-0.1]# main           &lt;==直接输入指令可执行！
[root@study main-0.1]# make uninstall &lt;==移除此软件！
rm -f /usr/local/bin/main
```

很有趣的练习吧！所以你只要下载 patch file 就能够对你的软件源代码更新了！只不过更新了源代码并非软件就更新！你还是得要将该软件进行编译后，才会是最终正确的软件喔！ 因为 patch 的功能主要仅只是更新源代码文件而已！切记切记！此外，如果你 patch 错误呢？没关系的！我们的 patch 是可以还原的啊！通过“ patch -R < ../main_0.1_to_0.2.patch ”就可以还原啦！很有趣吧！

例题：如果我有一个很旧版的软件，这个软件已经更新到很新的版本，例如核心，那么我可以使用 patch file 来更新吗？答：这个问题挺有趣的，首先，你必须要确定旧版本与新版本之间“确实有释出 patch file ”才行，以 kernel 2.2.xx 及 2.4.xx 来说，这两者基本上的架构已经不同了，所以两者间是无法以 patch file 来更新的。不过， 2.4.xx 与 2.4.yy 就可以更新了。不过，因为 kernel 每次推出的 patch 文件都仅针对前一个版本而已，所以假设要由 kernel 2.4.20 升级到 2.4.26 ，就必须要使用 patch 2.4.21, 2.4.22, 2.4.23, 2.4.24, 2.4.25, 2.4.26 六个文件来“依序更新”才行喔！当然，如果有朋友帮你比对过 2.4.20 与 2.4.26 ，那你自然就可以使用该 patch file 来直接一次更新啰！

21.5 函数库管理

在我们的 Linux 操作系统当中，函数库是很重要的一个项目。 因为很多的软件之间都会互相取用彼此提供的函数库来进行特殊功能的运行， 例如很多需要验证身份的程序都习惯利用 PAM 这个模块提供的验证机制来实作，而很多网络连线机制则习惯利用 SSL 函数库来进行连线加密的机制。所以说，函数库的利用是很重要的。不过， 函数库又依照是否被编译到程序内部而分为动态与静态函数库，这两者之间有何差异？哪一种函数库比较好？ 下面我们就来谈一谈先！

21.5.1 动态与静态函数库

首先我们要知道的是，函数库的类型有哪些？依据函数库被使用的类型而分为两大类，分别是静态 （Static） 与动态 （Dynamic） 函数库两类。下面我们来谈一谈这两种类行的函数库吧！

- 静态函数库的特色：
- 扩展名：（扩展名为 .a） 这类的函数库通常扩展名为 libxxx.a 的类型；
- 编译行为： 这类函数库在编译的时候会直接整合到执行程序当中，所以利用静态函数库编译成的文件会比较大一些喔；
- 独立执行的状态： 这类函数库最大的优点，就是编译成功的可可执行文件可以独立执行，而不需要再向外部要求读取函数库的内容 （请参照动态函数库的说明）。
- 升级难易度： 虽然可执行文件可以独立执行，但因为函数库是直接整合到可执行文件中， 因此若函数库升级时，整个可执行文件必须要重新编译才能将新版的函数库整合到程序当中。 也就是说，在升级方面，只要函数库升级了，所有将此函数库纳入的程序都需要重新编译！
- 动态函数库的特色：
- 扩展名：（扩展名为 .so） 这类函数库通常扩展名为 libxxx.so 的类型；
- 编译行为： 动态函数库与静态函数库的编译行为差异挺大的。 与静态函数库被整个捉到程序中不同的，动态函数库在编译的时候，在程序里面只有一个“指向 （Pointer）”的位置而已。也就是说，动态函数库的内容并没有被整合到可执行文件当中，而是当可执行文件要使用到函数库的机制时， 程序才会去读取函数库来使用。由于可执行文件当中仅具有指向动态函数库所在的指标而已， 并不包含函数库的内容，所以他的文件会比较小一点。

- 独立执行的状态： 这类型的函数库所编译出来的程序不能被独立执行， 因为当我们使用到函数库的机制时，程序才会去读取函数库，所以函数库文件“必须要存在”才行，而且，函数库的“所在目录也不能改变”，因为我们的可可执行文件里面仅有“指标”亦即当要取用该动态函数库时， 程序会主动去某个路径下读取，呵呵！所以动态函数库可不能随意移动或删除，会影响很多相依的程序软件喔！
- 升级难易度： 虽然这类型的可执行文件无法独立运行，然而由于是具有指向的功能， 所以，当函数库升级后，可执行文件根本不需要进行重新编译的行为，因为可执行文件会直接指向新的函数库文件 （前提是函数库新旧版本的文件名相同喔！）。

目前的 Linux distribution 比较倾向于使用动态函数库，因为如同上面提到的最重要的一点，就是函数库的升级方便！由于 Linux 系统里面的软件相依性太复杂了，如果使用太多的静态函数库，那么升级某一个函数库时， 都会对整个系统造成很大的冲击！因为其他相依的可执行文件也要同时重新编译啊！ 这个时候动态函数库可就有用多了，因为只要动态函数库升级就好，其他的软件根本无须变动。

那么这些函数库放置在哪里呢？绝大多数的函数库都放置在：/lib64, /lib 目录下！ 此外，Linux 系统里面很多的函数库其实 kernel 就提供了，那么 kernel 的函数库放在哪里？呵呵！就是在 /lib/modules 里面啦！里面的数据可多着呢！不过要注意的是， 不同版本的核心提供的函数库差异性是挺大的，所以 kernel 2.4.xx 版本的系统不要想将核心换成 2.6.xx 喔！ 很容易由于函数库的不同而导致很多原本可以执行的软件无法顺利运行呢！

21.5.2 ldconfig 与 /etc/ld.so.conf

在了解了动态与静态函数库，也知道我们目前的 Linux 大多是将函数库做成动态函数库之后，再来要知道的就是，那有没有办法增加函数库的读取性能？ 我们知道内存的存取速度是硬盘的好几倍，所以，如果我们将常用到的动态函数库先载入内存当中 （高速缓存, cache），如此一来，当软件要取用动态函数库时，就不需要从头由硬盘里面读出啰！ 这样不就可以增进动态函数库的读取速度？没错，是这样的！这个时候就需要 ldconfig 与 /etc/ld.so.conf 的协助了。

如何将动态函数库载入高速缓存内存当中呢？

1. 首先，我们必须要在 /etc/ld.so.conf 里面写下“ 想要读入高速缓存内存当中的动态函数库所在的目录”，注意喔， 是目录而不是文件；
2. 接下来则是利用 ldconfig 这个可执行文件将 /etc/ld.so.conf 的数据读入高速缓存当中；
3. 同时也将数据记录一份在 /etc/ld.so.cache 这个文件当中呐！

图21.5.1、使用 ldconfig 预载入动态函数库到内存中

事实上， ldconfig 还可以用来判断动态函数库的链接信息呢！赶紧利用 CentOS 来测试看看。假设你想要将目前你系统下的 mariadb 函数库加入到高速缓存当中时，可以这样做：

```
[root@study ~]# ldconfig [-f conf] [ -C cache]
[root@study ~]# ldconfig [-p]
选项与参数：
-f conf ：那个 conf 指的是某个文件名称，也就是说，使用 conf 作为 libarary
      函数库的取得路径，而不以 /etc/ld.so.conf 为默认值
-C cache：那个 cache 指的是某个文件名称，也就是说，使用 cache 作为高速缓存暂存
      的函数库数据，而不以 /etc/ld.so.cache 为默认值
-p    ：列出目前有的所有函数库数据内容 （在 /etc/ld.so.cache 内的数据！）

范例一：假设我的 Mariadb 数据库函数库在 /usr/lib64/mysql 当中，如何读进 cache ？
[root@study ~]# vim /etc/ld.so.conf.d/vbird.conf
/usr/lib64/mysql   &lt;==这一行新增的啦！

[root@study ~]# ldconfig  &lt;==画面上不会显示任何的信息，不要太紧张！正常的！

[root@study ~]# ldconfig -p
924 libs found in cache `/etc/ld.so.cache'
        p11-kit-trust.so （libc6,x86-64） =&gt; /lib64/p11-kit-trust.so
        libzapojit-0.0.so.0 （libc6,x86-64） =&gt; /lib64/libzapojit-0.0.so.0
....（下面省略）....
#       函数库名称 =&gt; 该函数库实际路径
```

通过上面的动作，我们可以将 Mariadb 的相关函数库给他读入高速缓存当中，这样可以加快函数库读取的效率呢！ 在某些时候，你可能会自行加入某些 Tarball 安装的动态函数库，而你想要让这些动态函数库的相关链接可以被读入到高速缓存当中， 这个时候你可以将动态函数库所在的目录名称写入 /etc/ld.so.conf.d/yourfile.conf 当中，然后执行 ldconfig 就可以啦！

21.5.3 程序的动态函数库解析： ldd

说了这么多，那么我如何判断某个可执行的 binary 文件含有什么动态函数库呢？很简单，利用 ldd 就可以晓得了！例如我想要知道 /usr/bin/passwd 这个程序含有的动态函数库有哪些，可以这样做：

```
[root@study ~]# ldd [-vdr] [filename]
选项与参数：
-v ：列出所有内容信息；
-d ：重新将数据有遗失的 link 点秀出来！
-r ：将 ELF 有关的错误内容秀出来！

范例一：找出 /usr/bin/passwd 这个文件的函数库数据
[root@study ~]# ldd /usr/bin/passwd
....（前面省略）....
        libpam.so.0 =&gt; /lib64/libpam.so.0 （0x00007f5e683dd000）         &lt;==PAM
        libpam_misc.so.0 =&gt; /lib64/libpam_misc.so.0 （0x00007f5e681d8000）
        libaudit.so.1 =&gt; /lib64/libaudit.so.1 （0x00007f5e67fb1000）       &lt;==SELin
        libselinux.so.1 =&gt; /lib64/libselinux.so.1 （0x00007f5e67d8c000）    &lt;==SELin
....（下面省略）....
# 我们前言的部分不是一直提到 passwd 有使用到 pam 的模块吗！怎么知道？
# 利用 ldd 察看一下这个文件，看到 libpam.so 了吧？这就是 pam 提供的函数库

范例二：找出 /lib64/libc.so.6 这个函数的相关其他函数库！
[root@study ~]# ldd -v /lib64/libc.so.6
        /lib64/ld-linux-x86-64.so.2 （0x00007f7acc68f000）
        linux-vdso.so.1 =&gt;  （0x00007fffa975b000）

        Version information:  &lt;==使用 -v 选项，增加显示其他版本信息！
        /lib64/libc.so.6:
                ld-linux-x86-64.so.2 （GLIBC_2.3） =&gt; /lib64/ld-linux-x86-64.so.2
                ld-linux-x86-64.so.2 （GLIBC_PRIVATE） =&gt; /lib64/ld-linux-x86-64.so.2
```

未来如果你常常升级安装 RPM 的软件时 （下一章节会介绍），应该常常会发现那个“ 相依属性”的问题吧！没错！我们可以先以 ldd 来视察“相依函数库”之间的相关性！以先取得了解！例如上面的例子中，我们检查了 libc.so.6 这个在 /lib64 当中的函数库，结果发现他其实还跟 ld-linux-x86-64.so.2 有关！所以我们就需要来了解一下，那个文件到底是什么软件的函数库呀？使用 -v 这个参数还可以得知该函数库来自于哪一个软件！像上面的数据中，就可以得到该 libc.so.6 其实可以支持 GLIBC_2.3 等的版本！

21.6 检验软件正确性

前面提到很多升级与安装需要注意的事项，因为我们需要克服很多的程序漏洞，所以需要前往 Linux distribution 或者是某些软件开发商的网站，下载最新并且较安全的软件文件来安装才行。 好了，那么“有没有可能我们下载的文件本身就有问题？” 是可能的！因为 cracker 无所不在，很多的软件开发商已经公布过他们的网页所放置的文件曾经被窜改过！ 那怎么办？连下载原版的数据都可能有问题了？难道没有办法判断文件的正确性吗？

这个时候我们就要通过每个文件独特的指纹验证数据了！因为每个文件的内容与文件大小都不相同， 所以如果一个文件被修改之后，必然会有部分的信息不一样！利用这个特性，我们可以使用 MD5/sha1 或更严密的 sha256 等指纹验证机制来判断该文件有没有被更动过！举个例子来说，在每个 CentOS 7.x 原版光盘的下载点都会有提供几个特别的文件， 你可以先到下面的链接看看：

- http://ftp.ksu.edu.tw/FTP/CentOS/7/isos/x86_64/

仔细看喔，上述的 URL 里面除了有所有光盘的下载点之外，还有提供刚刚说到的 md5, sha1, sha256 等指纹验证机制喔！通过这个编码的比对， 我们就可以晓得下载的文件是否有问题。那么万一 CentOS 提供的光盘镜像文件被下载之后，让有心人士偷偷修改过，再转到 Internet 上面流传，那么你下载的这个文件偏偏不是原厂提供的，呵呵！ 你能保证该文件的内容完全没有问题吗？当然不能对不对！是的，这个时候就有 md5sum, sha1sum, sha256sum 这几文件指纹的咚咚出现啦！说说他的用法吧！

21.6.1 md5sum / sha1sum / sha256sum

目前有多种机制可以计算文件的指纹码，我们选择使用较为广泛的 MD5, SHA1 或 SHA256 加密机制来处理， 例如上面链接中 CentOS 7.x 的相关指纹确认。不过 ISO文件实在太大了，下载来确认实在很浪费带宽。 所以我们拿前一个小节谈到的 NTP 软件来检查看看好了。记得我们下载的 NTP 软件版本为 4.2.8p3 这一版， 在官网上面仅有提供 md5sum 的数据而已，在下载页面的 MD5 数据为：

```
b98b0cbb72f6df04608e1dd5f313808b  ntp-4.2.8p3.tar.gz
```

如何确认我们下载的文件是正确没问题的呢？这样处理一下：

```
[root@study ~]# md5sum/sha1sum/sha256sum [-bct] filename
[root@study ~]# md5sum/sha1sum/sha256sum [--status&#124;--warn] --check filename
选项与参数：
-b ：使用 binary 的读档方式，默认为 Windows/DOS 文件型态的读取方式；
-c ：检验文件指纹；
-t ：以文字体态来读取文件指纹。

范例一：将刚刚的文件下载后，测试看看指纹码
[root@study ~]# md5sum ntp-4.2.8p3.tar.gz
b98b0cbb72f6df04608e1dd5f313808b  ntp-4.2.8p3.tar.gz
# 看！显示的编码是否与上面相同呢？赶紧测试看看！
```

一般而言，每个系统里面的文件内容大概都不相同，例如你的系统中的 /etc/passwd 这个登陆信息档与我的一定不一样，因为我们的使用者与密码、 Shell 及主文件夹等大概都不相同，所以由 md5sum 这个文件指纹分析程序所自行计算出来的指纹表当然就不相同啰！

好了，那么如何应用这个东西呢？基本上，你必须要在你的 Linux 系统上为你的这些重要的文件进行指纹数据库的创建 （好像在做户口调查！），将下面这些文件创建数据库：

- /etc/passwd
- /etc/shadow （假如你不让使用者改密码了）
- /etc/group
- /usr/bin/passwd
- /sbin/rpcbind
- /bin/login （这个也很容易被骇！）
- /bin/ls
- /bin/ps
- /bin/top

这几个文件最容易被修改了！因为很多木马程序执行的时候，还是会有所谓的“执行序, PID”为了怕被 root 追查出来，所以他们都会修改这些检查调度的文件，如果你可以替这些文件创建指纹数据库 （就是使用 md5sum 检查一次，将该文件指纹记录下来，然后常常以 shell script 的方式由程序自行来检查指纹表是否不同了！），那么对于文件系统会比较安全啦！

21.7 重点回顾

- 源代码其实大多是纯文本文件，需要通过编译器的编译动作后，才能够制作出 Linux 系统能够认识的可执行的 binary file ；
- 开放源代码可以加速软件的更新速度，让软件性能更快、漏洞修补更实时；
- 在 Linux 系统当中，最标准的 C 语言编译器为 gcc ；
- 在编译的过程当中，可以借由其他软件提供的函数库来使用该软件的相关机制与功能；
- 为了简化编译过程当中的复杂的指令输入，可以借由 make 与 makefile 规则定义，来简化程序的更新、编译与链接等动作；
- Tarball 为使用 tar 与 gzip/bzip2/xz 压缩功能所打包与压缩的，具有源代码的文件；
- 一般而言，要使用 Tarball 管理 Linux 系统上的软件，最好需要 gcc, make, autoconfig, kernel source, kernel header 等前驱软件才行，所以在安装 Linux 之初，最好就能够选择 Software development 以及 kernel development 之类的群组；
- 函数库有动态函数库与静态函数库，动态函数库在升级上具有较佳的优势。动态函数库的扩展名为 *.so* 而静态则是 .a ；
- patch 的主要功能在更新源代码，所以更新源代码之后，还需要进行重新编译的动作才行；
- 可以利用 ldconfig 与 /etc/ld.so.conf /etc/ld.so.conf.d/*.conf 来制作动态函数库的链接与高速缓存！
- 通过 MD5/SHA1/SHA256 的编码可以判断下载的文件是否为原本厂商所释出的文件。

21.8 本章习题

实作题部分：

- 请前往企鹅游戏网站 http://xpenguins.seul.org/ 下载 xpenguins-2.2.tar.gz 源代码文件，并安装该软件。安装完毕之后，请在 GNOME 图形接口执行 xpenguins ，看看有没有出现如同官网上面出现的小企鹅？（你有可能需要安装 yum install libX*-devel 才行喔）

情境仿真题部分：

- 请依照下面的方式来创建你的系统的重要文件指纹码，并每日比对此重要工作。

  1. 将 /etc/{passwd,shadow,group} 以及系统上面所有的 SUID/SGID 文件创建文件列表，该列表文件名为“ important.file ”；

  ```
  [root@study ~]# ls /etc/{passwd,shadow,group} &gt; important.file
  [root@study ~]# find /usr/sbin /usr/bin -perm /6000 &gt;&gt; important.file
  ```

  2. 通过这个文件名列表，以名为 md5.checkfile.sh 的文件名去创建指纹码，并将该指纹码文件“ finger1.file ”设置成为不可修改的属性；

  ```
  [root@study ~]# vim md5.checkfile.sh
  #!/bin/bash
  for filename in $（cat important.file）
  do
          md5sum $filename &gt;&gt; finger1.file
  done

  [root@study ~]# sh md5.checkfile.sh
  [root@study ~]# chattr +i finger1.file
  ```

  3. 通过相同的机制去创建后续的分析数据为 finger_new.file ，并将两者进行比对，若有问题则提供 email 给 root 查阅：

```
[root@study ~]# vim md5.checkfile.sh
#!/bin/bash
if [ "$1" == "new" ]; then
    for filename in $（cat important.file）
    do
        md5sum $filename &gt;&gt; finger1.file
    done
    echo "New file finger1.file is created."
    exit 0
fi
if [ ! -f finger1.file ]; then
    echo "file: finger1.file NOT exist."
    exit 1
fi

[ -f finger_new.file ] && rm finger_new.file
for filename in $（cat important.file）
do
    md5sum $filename &gt;&gt; finger_new.file
done

testing=$（diff finger1.file finger_new.file）
if [ "$testing" != "" ]; then
    diff finger1.file finger_new.file &#124; mail -s 'finger trouble..' root
fi

[root@study ~]# vim /etc/crontab
30 2 * * * root cd /root; sh md5.checkfile.sh
```

如此一来，每天系统会主动的去分析你认为重要的文件之指纹数据，然后再加以分析，看看有没有被更动过。 不过，如果该变动是正常的，例如 CentOS 自动的升级时，那么你就得要删除 finger1.file ， 再重新创建一个新的指纹数据库才行！否则你会每天收到有问题信件的回报喔！

21.9 参考资料与延伸阅读

- [1]GNU 的 make 网页：http://www.gnu.org/software/make/manual/make.html
- 几种常见加密机制的全名： md5 （Message-Digest algorithm 5） http://en.wikipedia.org/wiki/MD5 sha （Secure Hash Algorithm） http://en.wikipedia.org/wiki/SHA_hash_functions des （Data Encryption Standard） http://en.wikipedia.org/wiki/Data_Encryption_Standard
- 洪朝贵老师的 C 程序语言：http://www.cyut.edu.tw/~ckhung/b/c/

2002/08/21：第一次完成 2003/02/11：重新编排与加入 FAQ 2004/03/25：原本是 Tarball 与 RPM ，本日开始将 Tarball 与 RPM 分开说明与讲解（后续会花好几天喔！），最重要的是 Source code 的说明，并提到相关的 gcc compile 功能等等！ 2004/04/10：经历了当兵中的无奈生活，终于将这篇给他完工了～（当时的鸟哥在将军渔港与青山港～） 2005/09/30：旧版文章 （Tarball 与 RPM 的简单说明） 移动到 此处 。 2005/10/01：将风格作个转变之外，也将一些测试移转到 FC4 上面进行！ 2008/01/10：感谢网友 ayttk 的说明，原本的 make 语法网页已经移动到其他地方了，请参考 这里。 2009/06/04：将基于 FC4 撰写的文章移动到 此处 2009/06/20：增加一个小练习，需要使用到 X software development 的软件群组喔！ 2009/09/15：加入一个情境仿真，其实有点功力练功练功而已的习题啰！

第二十二章、软件安装 RPM, SRPM 与 YUM

最近更新日期：20//

虽然使用源代码进行软件编译可以具有客制化的设置，但对于 Linux distribution 的发布商来说，则有软件管理不易的问题， 毕竟不是每个人都会进行源代码编译的。如果能够将软件预先在相同的硬件与操作系统上面编译好才发布的话， 不就能够让相同的 distribution 具有完全一致的软件版本吗？如果再加上简易的安装/移除/管理等机制的话， 对于软件控管就会简易的多。有这种东西吗？有的，那就是 RPM 与 YUM 这两个好用的咚咚。 既然这么好用，我们当然不能错过学习机会啰！赶紧来参详参详！

22.1 软件管理员简介

在前一章我们提到以源代码的方式来安装软件，也就是利用厂商释出的 Tarball 来进行软件的安装。不过，你应该很容易发现，那就是每次安装软件都需要侦测操作系统与环境、设置编译参数、实际的编译、 最后还要依据个人喜好的方式来安装软件到定位。这过程是真的很麻烦的，而且对于不熟整个系统的朋友来说，还真是累人啊！

那有没有想过，如果我的 Linux 系统与厂商的系统一模一样，那么在厂商的系统上面编译出来的可执行文件， 自然也就可以在我的系统上面跑啰！也就是说，厂商先在他们的系统上面编译好了我们使用者所需要的软件， 然后将这个编译好的可执行的软件直接释出给使用者来安装，如此一来，由于我们本来就使用厂商的 Linux distribution ，所以当然系统 （硬件与操作系统） 是一样的，那么使用厂商提供的编译过的可可执行文件就没有问题啦！ 说的比较白话一些，那就是利用类似 Windows 的安装方式，由程序开发者直接在已知的系统上面编译好，再将该程序直接给使用者来安装，如此而已。

那么如果在安装的时候还可以加上一些与这些程序相关的信息，将他创建成为数据库，那不就可以进行安装、反安装、 升级与验证等等的相关功能啰 （类似 Windows 下面的“新增移除程序”）？确实如此，在 Linux 上面至少就有两种常见的这方面的软件管理员，分别是 RPM 与 Debian 的 dpkg 。我们的 CentOS 主要是以 RPM 为主，但也不能不知道 dpkg 啦！所以下面就来约略介绍一下这两个玩意儿。

22.1.1 Linux 界的两大主流: RPM 与 DPKG

由于自由软件的蓬勃发展，加上大型 Unix-Like 主机的强大性能，让很多软件开发者将他们的软件使用 Tarball 来释出。 后来 Linux 发展起来后，由一些企业或社群将这些软件收集起来制作成为 distributions 以发布这好用的 Linux 操作系统。但后来发现到，这些 distribution 的软件管理实在伤脑筋， 如果软件有漏洞时，又该如何修补呢？使用 tarball 的方式来管理吗？又常常不晓得到底我们安装过了哪些程序？ 因此，一些社群与企业就开始思考 Linux 的软件管理方式。

如同刚刚谈过的方式，Linux 开发商先在固定的硬件平台与操作系统平台上面将需要安装或升级的软件编译好， 然后将这个软件的所有相关文件打包成为一个特殊格式的文件，在这个软件文件内还包含了预先侦测系统与相依软件的脚本， 并提供记载该软件提供的所有文件信息等。最终将这个软件文件释出。用户端取得这个文件后，只要通过特定的指令来安装， 那么该软件文件就会依照内部的脚本来侦测相依的前驱软件是否存在，若安装的环境符合需求，那就会开始安装， 安装完成后还会将该软件的信息写入软件管理机制中，以达成未来可以进行升级、移除等动作呢。

目前在 Linux 界软件安装方式最常见的有两种，分别是：

- dpkg： 这个机制最早是由 Debian Linux 社群所开发出来的，通过 dpkg 的机制，Debian 提供的软件就能够简单的安装起来，同时还能提供安装后的软件信息，实在非常不错。 只要是衍生于 Debian 的其他 Linux distributions 大多使用 dpkg 这个机制来管理软件的， 包括 B2D, Ubuntu 等等。
- RPM： 这个机制最早是由 Red Hat 这家公司开发出来的，后来实在很好用，因此很多 distributions 就使用这个机制来作为软件安装的管理方式。包括 Fedora, CentOS, SuSE 等等知名的开发商都是用这咚咚。

如前所述，不论 dpkg/rpm 这些机制或多或少都会有软件属性相依的问题，那该如何解决呢？其实前面不是谈到过每个软件文件都有提供相依属性的检查吗？那么如果我们将相依属性的数据做成列表， 等到实际软件安装时，若发生有相依属性的软件状况时，例如安装 A 需要先安装 B 与 C ，而安装 B 则需要安装 D 与 E 时，那么当你要安装 A ，通过相依属性列表，管理机制自动去取得 B, C, D, E 来同时安装， 不就解决了属性相依的问题吗？

没错！您眞聪明！目前新的 Linux 开发商都有提供这样的“线上升级”机制，通过这个机制， 原版光盘就只有第一次安装时需要用到而已，其他时候只要有网络，你就能够取得原本开发商所提供的任何软件了呢！ 在 dpkg 管理机制上就开发出 APT 的线上升级机制，RPM 则依开发商的不同，有 Red Hat 系统的 yum ， SuSE 系统的 Yast Online Update （YOU） 等。

distribution 代表	软件管理机制	使用指令	线上升级机制（指令）
Red Hat/Fedora	RPM	rpm, rpmbuild	YUM （yum）
Debian/Ubuntu	DPKG	dpkg	APT （apt-get）

我们这里使用的是 CentOS 系统嘛！所以说：使用的软件管理机制为 RPM 机制，而用来作为线上升级的方式则为 yum ！下面就让我们来谈谈 RPM 与 YUM 的相关说明吧！

22.1.2 什么是 RPM 与 SRPM

RPM 全名是“ RedHat Package Manager ”简称则为 RPM 啦！顾名思义，当初这个软件管理的机制是由 Red Hat 这家公司发展出来的。 RPM 是以一种数据库记录的方式来将你所需要的软件安装到你的 Linux 系统的一套管理机制。

他最大的特点就是将你要安装的软件先编译过， 并且打包成为 RPM 机制的包装文件，通过包装好的软件里头默认的数据库记录， 记录这个软件要安装的时候必须具备的相依属性软件，当安装在你的 Linux 主机时， RPM 会先依照软件里头的数据查询 Linux 主机的相依属性软件是否满足， 若满足则予以安装，若不满足则不予安装。那么安装的时候就将该软件的信息整个写入 RPM 的数据库中，以便未来的查询、验证与反安装！这样一来的优点是：

1. 由于已经编译完成并且打包完毕，所以软件传输与安装上很方便 （不需要再重新编译）；
2. 由于软件的信息都已经记录在 Linux 主机的数据库上，很方便查询、升级与反安装

但是这也造成些许的困扰。由于 RPM 文件是已经包装好的数据，也就是说， 里面的数据已经都“编译完成”了！所以，该软件文件几乎只能安装在原本默认的硬件与操作系统版本中。也就是说，你的主机系统环境必须要与当初创建这个软件文件的主机环境相同才行！ 举例来说，rp-pppoe 这个 ADSL 拨接软件，他必须要在 ppp 这个软件存在的环境下才能进行安装！如果你的主机并没有 ppp 这个软件，那么很抱歉，除非你先安装 ppp 否则 rp-pppoe 就是不让你安装的 （当然你可以强制安装，但是通常都会有点问题发生就是了！）。

所以，通常不同的 distribution 所释出的 RPM 文件，并不能用在其他的 distributions 上。举例来说，Red Hat 释出的 RPM 文件，通常无法直接在 SuSE 上面进行安装的。更有甚者，相同 distribution 的不同版本之间也无法互通，例如 CentOS 6.x 的 RPM 文件就无法直接套用在 CentOS 7.x ！因此，这样可以发现这些软件管理机制的问题是：

1. 软件文件安装的环境必须与打包时的环境需求一致或相当；
2. 需要满足软件的相依属性需求；
3. 反安装时需要特别小心，最底层的软件不可先移除，否则可能造成整个系统的问题！

那怎么办？如果我真的想要安装其他 distributions 提供的好用的 RPM 软件文件时？ 呵呵！还好，还有 SRPM 这个东西！SRPM 是什么呢？顾名思义，他是 Source RPM 的意思，也就是这个 RPM 文件里面含有源代码哩！特别注意的是，这个 SRPM 所提供的软件内容“并没有经过编译”， 它提供的是源代码喔！

通常 SRPM 的扩展名是以 *.src.rpm 这种格式来命名的。不过，既然 SRPM 提供的是源代码，那么为什么我们不使用 Tarball 直接来安装就好了？这是因为 SRPM 虽然内容是源代码， 但是他仍然含有该软件所需要的相依性软件说明、以及所有 RPM 文件所提供的数据。同时，他与 RPM 不同的是，他也提供了参数配置文件 （就是 configure 与 makefile）。所以，如果我们下载的是 SRPM ，那么要安装该软件时，你就必须要：

- 先将该软件以 RPM 管理的方式编译，此时 SRPM 会被编译成为 RPM 文件；
- 然后将编译完成的 RPM 文件安装到 Linux 系统当中

怪了，怎么 SRPM 这么麻烦呐！还要重新编译一次，那么我们直接使用 RPM 来安装不就好了？通常一个软件在释出的时候，都会同时释出该软件的 RPM 与 SRPM 。我们现在知道 RPM 文件必须要在相同的 Linux 环境下才能够安装，而 SRPM 既然是源代码的格式，自然我们就可以通过修改 SRPM 内的参数配置文件，然后重新编译产生能适合我们 Linux 环境的 RPM 文件，如此一来，不就可以将该软件安装到我们的系统当中，而不必与原作者打包的 Linux 环境相同了？这就是 SRPM 的用处了！

文件格式	文件名格式	直接安装与否	内含程序类型	可否修改参数并编译
RPM	xxx.rpm	可	已编译	不可
SRPM	xxx.src.rpm	不可	未编译之源代码	可



Tips 为何说 CentOS 是“社群维护的企业版”呢？ Red Hat 公司的 RHEL 释出后，连带会将 SRPM 释出。 社群的朋友就将这些 SRPM 收集起来并重新编译成为所需要的软件，再重复释出成为 CentOS，所以才能号称与 Red Hat 的 RHEL 企业版同步啊！真要感谢 SRPM 哩！如果你想要理解 CentOS 是如何编译一支程序的， 也能够通过学习 SRPM 内含的编译参数，来学习的啊！

22.1.3 什么是 i386, i586, i686, noarch, x86_64

从上面的说明，现在我们知道 RPM 与 SRPM 的格式分别为：

```
xxxxxxxxx.rpm   &lt;==RPM 的格式，已经经过编译且包装完成的 rpm 文件；
xxxxx.src.rpm   &lt;==SRPM的格式，包含未编译的源代码信息。
```

那么我们怎么知道这个软件的版本、适用的平台、编译释出的次数呢？只要通过文件名就可以知道了！例如 rp-pppoe-3.11-5.el7.x86_64.rpm 这的文件的意义为：

```
rp-pppoe -        3.11    -     5        .el7.x86_64  .rpm
软件名称     软件的版本信息 释出的次数 适合的硬件平台 扩展名
```

除了后面适合的硬件平台与扩展名外，主要是以“-”来隔开各个部分，这样子可以很清楚的发现该软件的名称、 版本信息、打包次数与操作的硬件平台！好了，来谈一谈每个不同的地方吧：

- 软件名称： 当然就是每一个软件的名称了！上面的范例就是 rp-pppoe 。
- 版本信息： 每一次更新版本就需要有一个版本的信息，否则如何知道这一版是新是旧？这里通常又分为主版本跟次版本。以上面为例，主版本为 3 ，在主版本的架构下更动部分源代码内容，而释出一个新的版本，就是次版本啦！以上面为例，就是 11 啰！所以版本名就为 3.11
- 释出版本次数： 通常就是编译的次数啦！那么为何需要重复的编译呢？这是由于同一版的软件中，可能由于有某些 bug 或者是安全上的顾虑，所以必须要进行小幅度的 patch 或重设一些编译参数。 设置完成之后重新编译并打包成 RPM 文件！因此就有不同的打包数出现了！
- 操作硬件平台： 这是个很好玩的地方，由于 RPM 可以适用在不同的操作平台上，但是不同的平台设置的参数还是有所差异性！ 并且，我们可以针对比较高阶的 CPU 来进行最优化参数的设置，这样才能够使用高阶 CPU 所带来的硬件加速功能。 所以就有所谓的 i386, i586, i686, x86_64 与 noarch 等的文件名称出现了！

```
| 平台名称 | 适合平台说明 |
| --- | --- |
| i386 | 几乎适用于所有的 x86 平台，不论是旧的 pentum 或者是新的 Intel Core 2 与 K8 系列的 CP
| i586 | 就是针对 586 等级的计算机进行最优化编译。那是哪些 CPU 呢？包括 pentum 第一代 MMX CPU，
| i686 | 在 pentun II 以后的 Intel 系列 CPU ，及 K7 以后等级的 CPU 都属于这个 686 等级！ 由于
| x86_64 | 针对 64 位的 CPU 进行最优化编译设置，包括 Intel 的 Core 2 以上等级 CPU ，以及 AMD
| noarch | 就是没有任何硬件等级上的限制。一般来说，这种类型的 RPM 文件，里面应该没有 binary prog
```

截至目前为止（2015），就算是旧的个人计算机系统，堪用与能用的设备大概都至少是 Intel Core 2 以上等级的计算机主机，泰半都是 64 位的系统了！ 因此目前 CentOS 7 仅推出 x86_64 的软件版本，并没有提供 i686 以下等级的软件了！如果你的系统还是很老旧的机器， 那才有可能不支持 64 位的 Linux 系统。此外，目前仅存的软件版本大概也只剩下 i686 及 x86_64 还有不分版本的 noarch 而已， i386 只有在某些很特别的软件上才看到的到啦！

受惠于目前 x86 系统的支持方面，新的 CPU 都能够执行旧型 CPU 所支持的软件，也就是说硬件方面都可以向下相容的， 因此最低等级的 i386 软件可以安装在所有的 x86 硬件平台上面，不论是 32 位还是 64 位。但是反过来说就不行了。举例来说，目前硬件大多是 64 位的等级，因此你可以在该硬件上面安装 x86_64 或 i386 等级的 RPM 软件。但在你的旧型主机，例如 P-III/P-4 32 位机器上面，就不能够安装 x86_64 的软件！

根据上面的说明，其实我们只要选择 i686 版本来安装在你的 x86 硬件上面就肯定没问题。但是如果强调性能的话， 还是选择搭配你的硬件的 RPM 文件吧！毕竟该软件才有针对你的 CPU 硬件平台进行过参数最优化的编译嘛！

22.1.4 RPM 的优点

由于 RPM 是通过预先编译并打包成为 RPM 文件格式后，再加以安装的一种方式，并且还能够进行数据库的记载。 所以 RPM 有以下的优点：

- RPM 内含已经编译过的程序与配置文件等数据，可以让使用者免除重新编译的困扰；
- RPM 在被安装之前，会先检查系统的硬盘容量、操作系统版本等，可避免文件被错误安装；
- RPM 文件本身提供软件版本信息、相依属性软件名称、软件用途说明、软件所含文件等信息，便于了解软件；
- RPM 管理的方式使用数据库记录 RPM 文件的相关参数，便于升级、移除、查询与验证。

为什么 RPM 在使用上很方便呢？我们前面提过， RPM 这个软件管理员所处理的软件，是由软件提供者在特定的 Linux 作业平台上面将该软件编译完成并且打包好。那使用者只要拿到这个打包好的软件， 然后将里头的文件放置到应该要摆放的目录，不就完成安装啰？对啦！就是这样！

但是有没有想过，我们在前一章里面提过的，有些软件是有相关性的，例如要安装网卡驱动程序，就得要有 kernel source 与 gcc 及 make 等软件。那么我们的 RPM 软件是否一定可以安装完成呢？如果该软件安装之后，却找不到他相关的前驱软件， 那不是挺麻烦的吗？因为安装好的软件也无法使用啊！

为了解决这种具有相关性的软件之间的问题 （就是所谓的软件相依属性），RPM 就在提供打包的软件时，同时加入一些讯息登录的功能，这些讯息包括软件的版本、 打包软件者、相依属性的其他软件、本软件的功能说明、本软件的所有文件记录等等，然后在 Linux 系统上面亦创建一个 RPM 软件数据库，如此一来，当你要安装某个以 RPM 型态提供的软件时，在安装的过程中， RPM 会去检验一下数据库里面是否已经存在相关的软件了， 如果数据库显示不存在，那么这个 RPM 文件“默认”就不能安装。呵呵！没有错，这个就是 RPM 类型的文件最为人所诟病的“软件的属性相依”问题啦！

22.1.5 RPM 属性相依的克服方式： YUM 线上升级

为了重复利用既有的软件功能，因此很多软件都会以函数库的方式释出部分功能，以方便其他软件的调用应用， 例如 PAM 模块的验证功能。此外，为了节省使用者的数据量，目前的 distributions 在释出软件时， 都会将软件的内容分为一般使用与开发使用 （development） 两大类。所以你才会常常看到有类似 pam-x.x.rpm 与 pam-devel-x.x.rpm 之类的文件名啊！而默认情况下，大部分的 software-devel-x.x.rpm 都不会安装，因为终端用户大部分不会去开发软件嘛！

因为有上述的现象，因此 RPM 软件文件就会有所谓的属性相依的问题产生 （其实所有的软件管理几乎都有这方面的情况存在）。 那有没有办法解决啊？前面不是谈到 RPM 软件文件内部会记录相依属性的数据吗？那想一想，要是我将这些相依属性的软件先列表， 在有要安装软件需求的时候，先到这个列表去找，同时与系统内已安装的软件相比较，没安装到的相依软件就一口气同时安装起来， 那不就解决了相依属性的问题了吗？有没有这种机制啊？有啊！那就是 YUM 机制的由来！

CentOS （1）先将释出的软件放置到 YUM 服务器内，然后（2）分析这些软件的相依属性问题，将软件内的记录信息写下来 （header）。 然后再将这些信息分析后记录成软件相关性的清单列表。这些列表数据与软件所在的本机或网络位置可以称呼为容器或软件仓库或软件库 （repository）。 当用户端有软件安装的需求时，用户端主机会主动的向网络上面的 yum 服务器的软件库网址下载清单列表， 然后通过清单列表的数据与本机 RPM 数据库已存在的软件数据相比较，就能够一口气安装所有需要的具有相依属性的软件了。 整个流程可以简单的如下图说明：

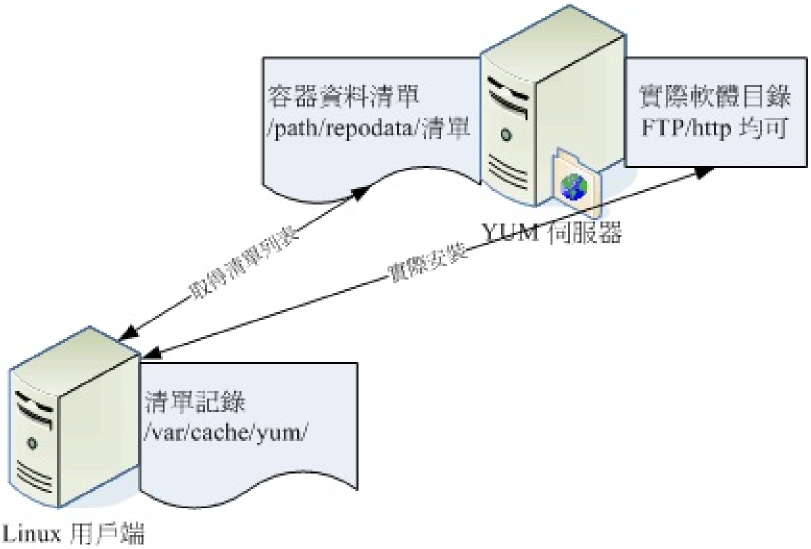


图22.1.1、YUM 使用的流程示意



图

Tips 所以软件仓库内的清单会记载每个文件的相依属性关系，以及所有文件的网络位置（URL）！由于记录了详细的软件网络位置， 所以有需要的时候，当然就会自动的从网络下载该软件啰！

当用户端有升级、安装的需求时， yum 会向软件库要求清单的更新，等到清单更新到本机的 /var/cache/yum 里面后， 等一下更新时就会用这个本机清单与本机的 RPM 数据库进行比较，这样就知道该下载什么软件。接下来 yum 会跑到软件库服务器 （yum server） 下载所需要的软件 （因为有记录软件所在的网址），然后再通过 RPM 的机制开始安装软件啦！这就是整个流程！ 谈到最后，还是需要动到 RPM 的啦！所以下个小节就让我们来谈谈 RPM 这咚咚吧！



Tips 为什么要做出“软件库”呢？由于 yum 服务器提供的 RPM 文件内容可能有所差异，举例来说，原厂释出的数据有 （1）原版数据； （2）更新数据 （update）； （3）特殊数据（例如第三方协力软件，或某些特殊功能的软件）。 这些软件文件基本上不会放置到一起，那如何分辨这些软件功能呢？就用“软件库”的概念来处理的啦！ 不同的“软件库”网址，可以放置不同的功能的软件之意！

22.2 RPM 软件管理程序： rpm

RPM 的使用其实不难，只要使用 rpm 这个指令即可！鸟哥最喜欢的就是 rpm 指令的查询功能了，可以让我很轻易的就知道某个系统有没有安装鸟哥要的软件呢！此外， 我们最好还是得要知道一下，到底 RPM 类型的文件他们是将软件的相关文件放置在哪里呢？还有，我们说的那个 RPM 的数据库又是放置在哪里呢？



Tips 事实上，下一小节要讲的 yum 就可以直接用来进行安装的动作，基本上 rpm 这个指令真的就只剩下查询与检验的功能啰！ 所以，查询与检验还是要学的，至于安装，通过 yum 就好了！

22.2.1 RPM 默认安装的路径

一般来说，RPM 类型的文件在安装的时候，会先去读取文件内记载的设置参数内容，然后将该数据用来比对 Linux 系统的环境，以找出是否有属性相依的软件尚未安装的问题。例如 Openssh 这个连线软件需要通过 Openssl 这个加密软件的帮忙，所以得先安装 openssl 才能装 openssh 的意思。那你的环境如果没有 openssl ， 你就无法安装 openssh 的意思啦。

若环境检查合格了，那么 RPM 文件就开始被安装到你的 Linux 系统上。安装完毕后，该软件相关的信息就会被写入 /var/lib/rpm/ 目录下的数据库文件中了。 上面这个目录内的数据很重要喔！因为未来如果我们有任何软件升级的需求，版本之间的比较就是来自于这个数据库，而如果你想要查询系统已经安装的软件，也是从这里查询的！同时，目前的 RPM 也提供数码签章信息， 这些数码签章也是在这个目录内记录的呢！所以说，这个目录得要注意不要被删除了啊！

那么软件内的文件到底是放置到哪里去啊？当然与文件系统有关对吧！我们在第五章的目录配置谈过每个目录的意义， 这里再次的强调啰：

/etc	一些配置文件放置的目录，例如 /etc/crontab
/usr/bin	一些可可执行文件案
/usr/lib	一些程序使用的动态函数库
/usr/share/doc	一些基本的软件使用手册与说明文档
/usr/share/man	一些 man page 文件

好了，下面我们就来针对每个 RPM 的相关指令来进行说明啰！

22.2.2 RPM 安装 （install）

因为安装软件是 root 的工作，因此你得要是 root 的身份才能够操作 rpm 这指令的。 用 rpm 来安装很简单啦！假设我要安装一个文件名为 rp-pppoe-3.11-5.el7.x86_64.rpm 的文件，那么我可以这样：（假设原版光盘已经放在 /mnt 下面了）

```
[root@study ~]# rpm -i /mnt/Packages/rp-pppoe-3.11-5.el7.x86_64.rpm
```

不过，这样的参数其实无法显示安装的进度，所以，通常我们会这样下达安装指令：

```
[root@study ~]# rpm -ivh package_name
选项与参数：
-i ：install 的意思
-v ：察看更细部的安装信息画面
-h ：以安装信息列显示安装进度

范例一：安装原版光盘上的 rp-pppoe 软件
[root@study ~]# rpm -ivh /mnt/Packages/rp-pppoe-3.11-5.el7.x86_64.rpm
Preparing...                          ################################# [100%]
Updating / installing...
   1:rp-pppoe-3.11-5.el7              ################################# [100%]

范例二、一口气安装两个以上的软件时：
[root@study ~]# rpm -ivh a.i386.rpm b.i386.rpm *.rpm
# 后面直接接上许多的软件文件！

范例三、直接由网络上面的某个文件安装，以网址来安装：
[root@study ~]# rpm -ivh http://website.name/path/pkgname.rpm
```

另外，如果我们在安装的过程当中发现问题，或者已经知道会发生的问题， 而还是“执意”要安装这个软件时，可以使用如下的参数“强制”安装上去：

rpm 安装时常用的选项与参数说明

可下达的选项	代表意义
--nodeps	使用时机：当发生软件属性相依问题而无法安装，但你执意安装时 危险性： 软件会有相依性的原因是因为彼此会使用到对方的机制或功能，如果强制安装而不考虑软件的属性相依， 则可能会造成该软件的无法正常使用！
--replacefiles	使用时机： 如果在安装的过程当中出现了“某个文件已经被安装在你的系统上面”的信息，又或许出现版本不合的讯息 （confilcting files） 时，可以使用这个参数来直接覆盖文件。危险性： 覆盖的动作是无法复原的！所以，你必须要很清楚的知道被覆盖的文件是真的可以被覆盖喔！否则会欲哭无泪！
--replacepkgs	使用时机： 重新安装某个已经安装过的软件！如果你要安装一堆 RPM 软件文件时，可以使用 rpm -ivh *.rpm ，但若某些软件已经安装过了， 此时系统会出现“某软件已安装”的信息，导致无法继续安装。此时可使用这个选项来重复安装喔！
--force	使用时机：这个参数其实就是 --replacefiles 与 --replacepkgs 的综合体！
--test	使用时机： 想要测试一下该软件是否可以被安装到使用者的 Linux 环境当中，可找出是否有属性相依的问题。范例为： `rpm -ivh pkgname.i386.rpm --test`
--justdb	使用时机： 由于 RPM 数据库破损或者是某些缘故产生错误时，可使用这个选项来更新软件在数据库内的相关信息。
--nosignature	使用时机： 想要略过数码签章的检查时，可以使用这个选项。
--prefix 新路径	使用时机： 要将软件安装到其他非正规目录时。举例来说，你想要将某软件安装到 /usr/local 而非正规的 /bin, /etc 等目录， 就可以使用“ --prefix /usr/local ”来处理了。
--noscripts	使用时机：不想让该软件在安装过程中自行执行某些系统指令。说明：RPM 的优点除了可以将文件放置到定位之外，还可以自动执行一些前置作业的指令，例如数据库的初始化。 如果你不想要让 RPM 帮你自动执行这一类型的指令，就加上他吧！

一般来说，rpm 的安装选项与参数大约就是这些了。通常鸟哥建议直接使用 -ivh 就好了， 如果安装的过程中发现问题，一个一个去将问题找出来，尽量不要使用“ 暴力安装法 ”，就是通过 --force 去强制安装！ 因为可能会发生很多不可预期的问题呢！除非你很清楚的知道使用上面的参数后，安装的结果是你预期的！

例题：在没有网络的前提下，你想要安装一个名为 pam-devel 的软件，你手边只有原版光盘，该如何是好？答：你可以通过挂载原版光盘来进行数据的查询与安装。请将原版光盘放入光驱，下面我们尝试将光盘挂载到 /mnt 当中， 并据以处理软件的下载啰：

- 挂载光盘，使用： mount /dev/sr0 /mnt
- 找出文件的实际路径：find /mnt -name 'pam-devel*'
- 测试此软件是否具有相依性： rpm -ivh pam-devel... --test
- 直接安装： rpm -ivh pam-devel...

- 卸载光盘： umount /mnt

在鸟哥的系统中，刚好这个软件并没有属性相依的问题，因此最后一个步骤可以顺利的进行下去呢！

22.2.3 RPM 升级与更新 （upgrade/freshen）

使用 RPM 来升级真是太简单了！就以 -Uvh 或 -Fvh 来升级即可，而 -Uvh 与 -Fvh 可以用的选项与参数，跟 install 是一样的。不过， -U 与 -F 的意义还是不太一样的，基本的差别是这样的：

-Uvh	后面接的软件即使没有安装过，则系统将予以直接安装； 若后面接的软件有安装过旧版，则系统自动更新至新版；
-Fvh	如果后面接的软件并未安装到你的 Linux 系统上，则该软件不会被安装；亦即只有已安装至你 Linux 系统内的软件会被“升级”！

由上面的说明来看，如果你想要大量的升级系统旧版本的软件时，使用 -Fvh 则是比较好的作法，因为没有安装的软件才不会被不小心安装进系统中。但是需要注意的是，如果你使用的是 -Fvh ，偏偏你的机器上尚无这一个软件，那么很抱歉，该软件并不会被安装在你的 Linux 主机上面，所以请重新以 ivh 来安装吧！

早期没有 yum 的环境下面，同时网络带宽也很糟糕的状况下，通常有的朋友在进行整个操作系统的旧版软件修补时，喜欢这么进行：

1. 先到各发展商的 errata 网站或者是国内的 FTP 图像站捉下来最新的 RPM 文件；
2. 使用 -Fvh 来将你的系统内曾安装过的软件进行修补与升级！（真是方便呀！）

所以，在不晓得 yum 功能的情况下，你依旧可以到 CentOS 的映设站台下载 updates 数据，然后利用上述的方法来一口气升级！ 当然啰，升级也是可以利用 --nodeps/--force 等等的参数啦！ 不过，现在既然有 yum 的机制在，这个笨方法当然也就不再需要了！

22.2.4 RPM 查询 （query）

RPM 在查询的时候，其实查询的地方是在 /var/lib/rpm/ 这个目录下的数据库文件啦！另外，RPM 也可以查询未安装的 RPM 文件内的信息喔！那如何去查询呢？ 我们先来谈谈可用的选项有哪些？

```
[root@study ~]# rpm -qa                              &lt;==已安装软件
[root@study ~]# rpm -q[licdR] 已安装的软件名称       &lt;==已安装软件
[root@study ~]# rpm -qf 存在于系统上面的某个文件名     &lt;==已安装软件
[root@study ~]# rpm -qp[licdR] 未安装的某个文件名称  &lt;==查阅RPM文件
选项与参数：
查询已安装软件的信息：
-q  ：仅查询，后面接的软件名称是否有安装；
-qa ：列出所有的，已经安装在本机 Linux 系统上面的所有软件名称；
-qi ：列出该软件的详细信息 （information），包含开发商、版本与说明等；
-ql ：列出该软件所有的文件与目录所在完整文件名 （list）；
-qc ：列出该软件的所有配置文件 （找出在 /etc/ 下面的文件名而已）
-qd ：列出该软件的所有说明文档 （找出与 man 有关的文件而已）
-qR ：列出与该软件有关的相依软件所含的文件 （Required 的意思）
-qf ：由后面接的文件名称，找出该文件属于哪一个已安装的软件；
-q --scripts：列出是否含有安装后需要执行的脚本档，可用以 debug 喔！
查询某个 RPM 文件内含有的信息：
-qp[icdlR]：注意 -qp 后面接的所有参数以上面的说明一致。但用途仅在于找出
        某个 RPM 文件内的信息，而非已安装的软件信息！注意！
```

在查询的部分，所有的参数之前都需要加上 -q 才是所谓的查询！查询主要分为两部分， 一个是查已安装到系统上面的的软件信息，这部份的信息都是由 /var/lib/rpm/ 所提供。另一个则是查某个 rpm 文件内容， 等于是由 RPM 文件内找出一些要写入数据库内的信息就是了，这部份就得要使用 -qp （p 是 package 的意思）。 那就来看看几个简单的范例吧！

```
范例一：找出你的 Linux 是否有安装 logrotate 这个软件？
[root@study ~]# rpm -q logrotate
logrotate-3.8.6-4.el7.x86_64
[root@study ~]# rpm -q logrotating
package logrotating is not installed
# 注意到，系统会去找是否有安装后面接的软件名称。注意，不必要加上版本喔！
# 至于显示的结果，一看就知道有没有安装啦！

范例二：列出上题当中，属于该软件所提供的所有目录与文件：
[root@study ~]# rpm -ql logrotate
/etc/cron.daily/logrotate
/etc/logrotate.conf
....（以下省略）....
# 可以看出该软件到底提供了多少的文件与目录，也可以追踪软件的数据。

范例三：列出 logrotate 这个软件的相关说明数据：
[root@study ~]# rpm -qi logrotate
Name        : logrotate                          # 软件名称
Version     : 3.8.6                              # 软件的版本
Release     : 4.el7                              # 释出的版本
Architecture: x86_64                             # 编译时所针对的硬件等级
Install Date: Mon 04 May 2015 05:52:36 PM CST    # 这个软件安装到本系统的时间
Group       : System Environment/Base            # 软件是放再哪一个软件群组中
Size        : 102451                             # 软件的大小
License     : GPL+                               # 释出的授权方式
Signature   : RSA/SHA256, Fri 04 Jul 2014 11:34:56 AM CST, Key ID 24c6a8a7f4a80eb5
Source RPM  : logrotate-3.8.6-4.el7.src.rpm      # 这就是 SRPM 的文件名
Build Date  : Tue 10 Jun 2014 05:58:02 AM CST    # 软件编译打包的时间
Build Host  : worker1.bsys.centos.org            # 在哪一部主机上面编译的
Relocations : （not relocatable）
Packager    : CentOS BuildSystem &lt;http://bugs.centos.org&gt;
Vendor      : CentOS
URL         : https://fedorahosted.org/logrotate/
Summary     : Rotates, compresses, removes and mails system log files
Description :                                    # 这个是详细的描述！
The logrotate utility is designed to simplify the administration of
log files on a system which generates a lot of log files.  Logrotate
allows for the automatic rotation compression, removal and mailing of
log files.  Logrotate can be set to handle a log file daily, weekly,
monthly or when the log file gets to a certain size.  Normally,
logrotate runs as a daily cron job.

Install the logrotate package if you need a utility to deal with the
log files on your system.
# 列出该软件的 information （信息），里面的信息可多着呢，包括了软件名称、
# 版本、开发商、SRPM文件名称、打包次数、简单说明信息、软件打包者、
# 安装日期等等！如果想要详细的知道该软件的数据，用这个参数来了解一下

范例四：分别仅找出 logrotate 的配置文件与说明文档
[root@study ~]# rpm -qc logrotate
[root@study ~]# rpm -qd logrotate

范例五：若要成功安装 logrotate ，他还需要什么文件的帮忙？
[root@study ~]# rpm -qR logrotate
/bin/sh
config（logrotate） = 3.8.6-4.el7
coreutils &gt;= 5.92
....（以下省略）....
# 由这里看起来，呵呵～还需要很多文件的支持才行喔！

范例六：由上面的范例五，找出 /bin/sh 是那个软件提供的？
[root@study ~]# rpm -qf /bin/sh
bash-4.2.46-12.el7.x86_64
# 这个参数后面接的可是“文件”呐！不像前面都是接软件喔！
# 这个功能在查询系统的某个文件属于哪一个软件所有的。

范例七：假设我有下载一个 RPM 文件，想要知道该文件的需求文件，该如何？
[root@study ~]# rpm -qpR filename.i386.rpm
# 加上 -qpR ，找出该文件需求的数据！
```

常见的查询就是这些了！要特别说明的是，在查询本机上面的 RPM 软件相关信息时， 不需要加上版本的名称，只要加上软件名称即可！因为他会由 /var/lib/rpm 这个数据库里面去查询， 所以我们可以不需要加上版本名称。但是查询某个 RPM 文件就不同了，我们必须要列出整个文件的完整文件名才行～ 这一点朋友们常常会搞错。下面我们就来做几个简单的练习吧！

例题：

1. 我想要知道我的系统当中，以 c 开头的软件有几个，如何实做？
2. 我的 WWW 服务器为 Apache ，我知道他使用的 RPM 软件文件名为 httpd 。现在，我想要知道这个软件的所有配置文件放置在何处，可以怎么作？
3. 承上题，如果查出来的设置文件已经被我改过，但是我忘记了曾经修改过哪些地方，所以想要直接重新安装一次该软件，该如何作？
4. 如果我误砍了某个重要文件，例如 /etc/crontab，偏偏不晓得他属于哪一个软件，该怎么办？

答：

1. rpm -qa | grep ^c | wc -l
2. rpm -qc httpd
3. 假设该软件在网络上的网址为： http://web.site.name/path/httpd-x.x.xx.i386.rpm 则我可以这样做： rpm -ivh http://web.site.name/path/httpd-x.x.xx.i386.rpm --replacepkgs
4. 虽然已经没有这个文件了，不过没有关系，因为 RPM 有记录在 /var/lib/rpm 当中的数据库啊！所以直接下达： rpm -qf /etc/crontab 就可以知道是那个软件啰！重新安装一次该软件即可！

22.2.5 RPM 验证与数码签章 （Verify/signature）

验证 （Verify） 的功能主要在于提供系统管理员一个有用的管理机制！作用的方式是“使用 /var/lib/rpm 下面的数据库内容来比对目前 Linux 系统的环境下的所有软件文件 ”也就是说，当你有数据不小心遗失， 或者是因为你误杀了某个软件的文件，或者是不小心不知道修改到某一个软件的文件内容， 就用这个简单的方法来验证一下原本的文件系统吧！好让你了解这一阵子到底是修改到哪些文件数据了！验证的方式很简单：

```
[root@study ~]# rpm -Va
[root@study ~]# rpm -V  已安装的软件名称
[root@study ~]# rpm -Vp 某个 RPM 文件的文件名
[root@study ~]# rpm -Vf 在系统上面的某个文件
选项与参数：
-V  ：后面加的是软件名称，若该软件所含的文件被更动过，才会列出来；
-Va ：列出目前系统上面所有可能被更动过的文件；
-Vp ：后面加的是文件名称，列出该软件内可能被更动过的文件；
-Vf ：列出某个文件是否被更动过～

范例一：列出你的 Linux 内的 logrotate 这个软件是否被更动过？
[root@study ~]# rpm -V logrotate
# 如果没有出现任何讯息，恭喜你，该软件所提供的文件没有被更动过。
# 如果有出现任何讯息，才是有出现状况啊！

范例二：查询一下，你的 /etc/crontab 是否有被更动过？
[root@study ~]# rpm -Vf /etc/crontab
.......T.  c /etc/crontab
# 瞧！因为有被更动过，所以会列出被更动过的信息类型！
```

好了，那么我怎么知道到底我的文件被更动过的内容是什么？例如上面的范例二。呵呵！简单的说明一下吧！ 例如，我们检查一下 logrotate 这个软件：

```
[root@study ~]# rpm -ql logrotate
/etc/cron.daily/logrotate
/etc/logrotate.conf
/etc/logrotate.d
/usr/sbin/logrotate
/usr/share/doc/logrotate-3.8.6
/usr/share/doc/logrotate-3.8.6/CHANGES
/usr/share/doc/logrotate-3.8.6/COPYING
/usr/share/man/man5/logrotate.conf.5.gz
/usr/share/man/man8/logrotate.8.gz
/var/lib/logrotate.status
# 呵呵！共有 10 个文件啊！请修改 /etc/logrotate.conf 内的 rotate 变成 5

[root@study ~]# rpm -V logrotate
..5....T.  c /etc/logrotate.conf
```

你会发现在文件名之前有个 c ，然后就是一堆奇怪的文字了。那个 c 代表的是 configuration ， 就是配置文件的意思。至于最前面的几个信息是：

- S ：（file Size differs） 文件的容量大小是否被改变
- M ：（Mode differs） 文件的类型或文件的属性 （rwx） 是否被改变？如是否可执行等参数已被改变
- 5 ：（MD5 sum differs） MD5 这一种指纹码的内容已经不同
- D ：（Device major/minor number mis-match） 设备的主/次代码已经改变
- L ：（readLink（2） path mis-match） Link 路径已被改变
- U ：（User ownership differs） 文件的所属人已被改变
- G ：（Group ownership differs） 文件的所属群组已被改变
- T ：（mTime differs） 文件的创建时间已被改变
- P ：（caPabilities differ） 功能已经被改变

所以，如果当一个配置文件所有的信息都被更动过，那么他的显示就会是：

```
SM5DLUGTP c filename
```

至于那个 c 代表的是“ Config file ”的意思，也就是文件的类型，文件类型有下面这几类：

- c ：配置文件 （config file）
- d ：文件数据文件 （documentation）
- g ：鬼文件～通常是该文件不被某个软件所包含，较少发生！（ghost file）
- l ：授权文件 （license file）
- r ：读我文件 （read me）

经过验证的功能，你就可以知道那个文件被更动过。那么如果该文件的变更是“预期中的”，那么就没有什么大问题，但是如果该文件是“非预期的”，那么是否被入侵了呢？呵呵！得注意注意啰！ 一般来说，配置文件 （configure） 被更动过是很正常的，万一你的 binary program 被更动过呢？ 那就得要特别特别小心啊！



Tips 虽说家丑不可外扬，不过有件事情还是跟大家分享一下的好。鸟哥之前的主机曾经由于安装一套软件，导致被攻击成为跳板。 会发现的原因是系统中只要出现 *.patch 的扩展名时，使用 ls -l 就是显示不出来该文件名 （该文件名确实存在）。 找了好久，用了好多工具都找不出问题，最终利用 rpm -Va 找出来，原来好多 binary program 被更动过，连 init 都被恶搞！此时，赶紧重新安装 Linux 并移除那套软件，之后就比较正常了。所以说，这个 rpm -Va 是个好功能喔！

- 数码签章 （digital signature）

谈完了软件的验证后，不知道你有没有发现一个问题，那就是，验证只能验证软件内的信息与 /var/lib/rpm/ 里面的数据库信息而已，如果该软件文件所提供的数据本身就有问题，那你使用验证的手段也无法确定该软件的正确性啊！ 那如何解决呢？在 Tarball 与文件的验证方面，我们可以使用前一章谈到的 md5 指纹码来检查， 不过，连指纹码也可能会被窜改的嘛！那怎办？没关系，我们可以通过数码签章来检验软件的来源的！

就像你自己的签名一样，我们的软件开发商原厂所推出的软件也会有一个厂商自己的签章系统！ 只是这个签章被数码化了而已。厂商可以数码签章系统产生一个专属于该软件的签章，并将该签章的公钥 （public key） 释出。 当你要安装一个 RPM 文件时：

1. 首先你必须要先安装原厂释出的公钥文件；
2. 实际安装原厂的 RPM 软件时， rpm 指令会去读取 RPM 文件的签章信息，与本机系统内的签章信息比对，
3. 若签章相同则予以安装，若找不到相关的签章信息时，则给予警告并且停止安装喔。

我们 CentOS 使用的数码签章系统为 GNU 计划的 GnuPG （GNU Privacy Guard, GPG）[1]。 GPG 可以通过杂凑运算，算出独一无二的专属金钥系统或者是数码签章系统，有兴趣的朋友可以参考文末的延伸阅读， 去了解一下 GPG 加密的机制喔！这里我们仅简单的说明数码签章在 RPM 文件上的应用而已。 而根据上面的说明，我们也会知道首先必须要安装原厂释出的 GPG 数码签章的公钥文件啊！CentOS 的数码签章位于：

```
[root@study ~]# ll /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
-rw-r--r--. 1 root root 1690 Apr  1 06:27 /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
[root@study ~]# cat /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.5 （GNU/Linux）

mQINBFOn/0sBEADLDyZ+DQHkcTHDQSE0a0B2iYAEXwpPvs67cJ4tmhe/iMOyVMh9
....（中间省略）....
-----END PGP PUBLIC KEY BLOCK-----
```

从上面的输出，你会知道该数码签章码其实仅是一个乱数而已，这个乱数对于数码签章有意义而已， 我们看不懂啦！那么这个文件如何安装呢？通过下面的方式来安装即可喔！

```
[root@study ~]# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
```

由于不同版本 GPG 金钥文件放置的位置可能不同，不过文件名大多是以 GPG-KEY 来说明的， 因此你可以简单的使用 locate 或 find 来找寻，如以下的方式来搜寻即可：

```
[root@study ~]# locate GPG-KEY
[root@study ~]# find /etc -name '*GPG-KEY*'
```

那安装完成之后，这个金钥的内容会以什么方式呈现呢？基本上都是使用 pubkey 作为软件的名称的！ 那我们先列出金钥软件名称后，再以 -qi 的方式来查询看看该软件的信息为何：

```
[root@study ~]# rpm -qa &#124; grep pubkey
gpg-pubkey-f4a80eb5-53a7ff4b
[root@study ~]# rpm -qi gpg-pubkey-f4a80eb5-53a7ff4b
Name        : gpg-pubkey
Version     : f4a80eb5
Release     : 53a7ff4b
Architecture: （none）
Install Date: Fri 04 Sep 2015 11:30:46 AM CST
Group       : Public Keys
Size        : 0
License     : pubkey
Signature   : （none）
Source RPM  : （none）
Build Date  : Mon 23 Jun 2014 06:19:55 PM CST
Build Host  : localhost
Relocations : （not relocatable）
Packager    : CentOS-7 Key （CentOS 7 Official Signing Key） &lt;security@centos.org&gt;
Summary     : gpg（CentOS-7 Key （CentOS 7 Official Signing Key） &lt;security@centos.org&
Description :
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: rpm-4.11.1 （NSS-3）
....（下面省略）....
```

重点就是最后面出现的那一串乱码啦！那可是作为数码签章非常重要的一环哩！ 如果你忘记加上数码签章，很可能很多原版软件就不能让你安装啰～除非你利用 rpm 时选择略过数码签章的选项。

22.2.6 RPM 反安装与重建数据库 （erase/rebuilddb）

反安装就是将软件解除安装啦！要注意的是，“解安装的过程一定要由最上层往下解除”，以 rp-pppoe 为例，这一个软件主要是依据 ppp 这个软件来安装的，所以当你要解除 ppp 的时候，就必须要先解除 rp-pppoe 才行！否则就会发生结构上的问题啦！这个可以由建筑物来说明， 如果你要拆除五、六楼，那么当然要由六楼拆起，否则先拆的是第五楼时，那么上面的楼层难道会悬空？

移除的选项很简单，就通过 -e 即可移除。不过，很常发生软件属性相依导致无法移除某些软件的问题！ 我们以下面的例子来说明：

```
# 1\. 找出与 pam 有关的软件名称，并尝试移除 pam 这个软件：
[root@study ~]# rpm -qa &#124; grep pam
fprintd-pam-0.5.0-4.0.el7_0.x86_64
pam-1.1.8-12.el7.x86_64
gnome-keyring-pam-3.8.2-10.el7.x86_64
pam-devel-1.1.8-12.el7.x86_64
pam_krb5-2.4.8-4.el7.x86_64
[root@study ~]# rpm -e pam
error: Failed dependencies:  &lt;==这里提到的是相依性的问题
        libpam.so.0（）（64bit） is needed by （installed） systemd-libs-208-20.el7.x86_64
        libpam.so.0（）（64bit） is needed by （installed） libpwquality-1.2.3-4.el7.x86_64
....（以下省略）....

# 2\. 若仅移除 pam-devel 这个之前范例安装上的软件呢？
[root@study ~]# rpm -e pam-devel  &lt;==不会出现任何讯息！
[root@study ~]# rpm -q pam-devel
package pam-devel is not installed
```

从范例一我们知道 pam 所提供的函数库是让非常多其他软件使用的，因此你不能移除 pam ，除非将其他相依软件一口气也全部移除！你当然也能加 --nodeps 来强制移除， 不过，如此一来所有会用到 pam 函数库的软件，都将成为无法运行的程序，我想，你的主机也只好准备停机休假了吧！ 至于范例二中，由于 pam-devel 是依附于 pam 的开发工具，你可以单独安装与单独移除啦！

由于 RPM 文件常常会安装/移除/升级等，某些动作或许可能会导致 RPM 数据库 /var/lib/rpm/ 内的文件破损。果真如此的话，那你该如何是好？别担心，我们可以使用 --rebuilddb 这个选项来重建一下数据库喔！ 作法如下：

```
[root@study ~]# rpm --rebuilddb   &lt;==重建数据库
```

22.3 YUM 线上升级机制

我们在本章一开始的地方谈到过 yum 这玩意儿，这个 yum 是通过分析 RPM 的标头数据后，根据各软件的相关性制作出属性相依时的解决方案，然后可以自动处理软件的相依属性问题，以解决软件安装或移除与升级的问题。 详细的 yum 服务器与用户端之间的沟通，可以再回到前面的部分查阅一下图 22.1.1 的说明。

由于 distribution 必须要先释出软件，然后将软件放置于 yum 服务器上面，以提供用户端来要求安装与升级之用的。 因此我们想要使用 yum 的功能时，必须要先找到适合的 yum server 才行啊！而每个 yum server 可能都会提供许多不同的软件功能，那就是我们之前谈到的“软件库”啦！因此，你必须要前往 yum server 查询到相关的软件库网址后，再继续处理后续的设置事宜。

事实上 CentOS 在释出软件时已经制作出多部映射站台 （mirror site） 提供全世界的软件更新之用。 所以，理论上我们不需要处理任何设置值，只要能够连上 Internet ，就可以使用 yum 啰！下面就让我们来玩玩看吧！

22.3.1 利用 yum 进行查询、安装、升级与移除功能

yum 的使用真是非常简单，就是通过 yum 这个指令啊！那么这个指令怎么用呢？用法很简单，就让我们来简单的谈谈：

- 查询功能：yum [list|info|search|provides|whatprovides] 参数

如果想要查询利用 yum 来查询原版 distribution 所提供的软件，或已知某软件的名称，想知道该软件的功能， 可以利用 yum 相关的参数为：

```
[root@study ~]# yum [option] [查询工作项目] [相关参数]
选项与参数：
[option]：主要的选项，包括有：
  -y ：当 yum 要等待使用者输入时，这个选项可以自动提供 yes 的回应；
  --installroot=/some/path ：将该软件安装在 /some/path 而不使用默认路径
[查询工作项目] [相关参数]：这方面的参数有：
  search   ：搜寻某个软件名称或者是描述 （description） 的重要关键字；
  list     ：列出目前 yum 所管理的所有的软件名称与版本，有点类似 rpm -qa；
  info     ：同上，不过有点类似 rpm -qai 的执行结果；
  provides：从文件去搜寻软件！类似 rpm -qf 的功能！

范例一：搜寻磁盘阵列 （raid） 相关的软件有哪些？
[root@study ~]# yum search raid
Loaded plugins: fastestmirror, langpacks       # yum 系统自己找出最近的 yum server
Loading mirror speeds from cached hostfile     # 找出速度最快的那一部 yum server
 * base: ftp.twaren.net                        # 下面三个软件库，且来源为该服务器！
 * extras: ftp.twaren.net
 * updates: ftp.twaren.net
....（前面省略）....
dmraid-events-logwatch.x86_64 : dmraid logwatch-based email reporting
dmraid-events.x86_64 : dmevent_tool （Device-mapper event tool） and DSO
iprutils.x86_64 : Utilities for the IBM Power Linux RAID adapters
mdadm.x86_64 : The mdadm program controls Linux md devices （software RAID arrays）
....（后面省略）....
# 在冒号 （:） 左边的是软件名称，右边的则是在 RPM 内的 name 设置 （软件名）
```

```
# 瞧！上面的结果，这不就是与 RAID 有关的软件吗？如果想了解 mdadm 的软件内容呢？

范例二：找出 mdadm 这个软件的功能为何
[root@study ~]# yum info mdadm
Installed Packages       &lt;==这说明该软件是已经安装的了
Name        : mdadm      &lt;==这个软件的名称
Arch        : x86_64     &lt;==这个软件的编译架构
Version     : 3.3.2      &lt;==此软件的版本
Release     : 2.el7      &lt;==释出的版本
Size        : 920 k      &lt;==此软件的文件总容量
Repo        : installed  &lt;==软件库回报说已安装的
From repo   : anaconda
Summary     : The mdadm program controls Linux md devices （software RAID arrays）
URL         : http://www.kernel.org/pub/linux/utils/raid/mdadm/
License     : GPLv2+
Description : The mdadm program is used to create, manage, and monitor Linux MD （software
            : RAID） devices.  As such, it provides similar functionality to the raidtools
            : package.  However, mdadm is a single program, and it can perform
            : almost all functions without a configuration file, though a configuration
            : file can be used to help with some common tasks.
# 不要跟我说，上面说些啥？自己找字典翻一翻吧！拜托拜托！

范例三：列出 yum 服务器上面提供的所有软件名称
[root@study ~]# yum list
Installed Packages   &lt;==已安装软件
GConf2.x86_64                           3.2.6-8.el7                    @anaconda
LibRaw.x86_64                           0.14.8-5.el7.20120830git98d925 @base
ModemManager.x86_64                     1.1.0-6.git20130913.el7        @anaconda
....（中间省略）....
Available Packages   &lt;==还可以安装的其他软件
389-ds-base.x86_64                      1.3.3.1-20.el7_1               updates
389-ds-base-devel.x86_64                1.3.3.1-20.el7_1               updates
389-ds-base-libs.x86_64                 1.3.3.1-20.el7_1               updates
....（下面省略）....
# 上面提供的意义为：" 软件名称   版本   在那个软件库内 "

范例四：列出目前服务器上可供本机进行升级的软件有哪些？
[root@study ~]# yum list updates  &lt;==一定要是 updates 喔！
Updated Packages
NetworkManager.x86_64          1:1.0.0-16.git20150121.b4ea599c.el7_1       updates
NetworkManager-adsl.x86_64     1:1.0.0-16.git20150121.b4ea599c.el7_1       updates
....（下面省略）....
# 上面就列出在那个软件库内可以提供升级的软件与版本！

范例五：列出提供 passwd 这个文件的软件有哪些
[root@study ~]# yum provides passwd
passwd-0.79-4.el7.x86_64 : An utility for setting or changing passwords using PAM
Repo        : base

passwd-0.79-4.el7.x86_64 : An utility for setting or changing passwords using PAM
Repo        : @anaconda
# 找到啦！就是上面的这个软件提供了 passwd 这个程序！
```

通过上面的查询，你应该大致知道 yum 如何用在查询上面了吧？那么实际来应用一下：

例题：利用 yum 的功能，找出以 pam 为开头的软件名称有哪些？而其中尚未安装的又有哪些？答：可以通过如下的方法来查询：

```
[root@study ~]# yum list pam*
Installed Packages
pam.x86_64                      1.1.8-12.el7                 @anaconda
pam_krb5.x86_64                 2.4.8-4.el7                  @base
Available Packages &lt;==下面则是“可升级”的或“未安装”的
pam.i686                        1.1.8-12.el7_1.1             updates
pam.x86_64                      1.1.8-12.el7_1.1             updates
pam-devel.i686                  1.1.8-12.el7_1.1             updates
pam-devel.x86_64                1.1.8-12.el7_1.1             updates
pam_krb5.i686                   2.4.8-4.el7                  base
pam_pkcs11.i686                 0.6.2-18.el7                 base
pam_pkcs11.x86_64               0.6.2-18.el7                 base
```

如上所示，所以可升级者有 pam 这两个软件，完全没有安装的则是 pam-devel 等其他几个软件啰！

- 安装/升级功能：yum [install|update] 软件

既然可以查询，那么安装与升级呢？很简单啦！就利用 install 与 update 这两项工作来处理即可喔！

```
[root@study ~]# yum [option] [安装与升级的工作项目] [相关参数]
选项与参数：
  install ：后面接要安装的软件！
  update  ：后面接要升级的软件，若要整个系统都升级，就直接 update 即可

范例一：将前一个练习找到的未安装的 pam-devel 安装起来
[root@study ~]# yum install pam-devel
Loaded plugins: fastestmirror, langpacks    # 首先的 5 行在找出最快的 yum server
Loading mirror speeds from cached hostfile
 * base: ftp.twaren.net
 * extras: ftp.twaren.net
 * updates: ftp.twaren.net
Resolving Dependencies                      # 接下来先处理“属性相依”的软件问题
--&gt; Running transaction check
---&gt; Package pam-devel.x86_64 0:1.1.8-12.el7_1.1 will be installed
--&gt; Processing Dependency: pam（x86-64） = 1.1.8-12.el7_1.1 for package: pam-devel-
       1.1.8-12.el7_1.1.x86_64
--&gt; Running transaction check
---&gt; Package pam.x86_64 0:1.1.8-12.el7 will be updated
---&gt; Package pam.x86_64 0:1.1.8-12.el7_1.1 will be an update
--&gt; Finished Dependency Resolution
Dependencies Resolved

# 由上面的检查发现到 pam 这个软件也需要同步升级，这样才能够安装新版 pam-devel 喔！
# 至于下面则是一个总结的表格显示！
==========================================================================================
 Package              Arch             Version                     Repository         Size
==========================================================================================
Installing:
 pam-devel            x86_64           1.1.8-12.el7_1.1            updates           183 k
Updating for dependencies:
 pam                  x86_64           1.1.8-12.el7_1.1            updates           714 k

Transaction Summary
==========================================================================================
Install  1 Package                          # 要安装的是一个软件
Upgrade              （ 1 Dependent package）  # 因为相依属性问题，需要额外加装一个软件！

Total size: 897 k
Total download size: 183 k                  # 总共需要下载的容量！
Is this ok [y/d/N]: y   # 你得要自己决定是否要下载与安装！当然是 y 啊！
Downloading packages:                       # 开始下载啰！
warning: /var/cache/yum/x86_64/7/updates/packages/pam-devel-1.1.8-12.el7_1.1.x86_64.rpm:
         Header V3 RSA/SHA256 Signature, key ID f4a80eb5: NOKEY
```

```
Public key for pam-devel-1.1.8-12.el7_1.1.x86_64.rpm is not installed
pam-devel-1.1.8-12.el7_1.1.x86_64.rpm                              &#124; 183 kB  00:00:0
Retrieving key from file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
Importing GPG key 0xF4A80EB5:
 Userid     : "CentOS-7 Key （CentOS 7 Official Signing Key） &lt;security@centos.org&gt;'
 Fingerprint: 6341 ab27 53d7 8a78 a7c2 7bb1 24c6 a8a7 f4a8 0eb5
 Package    : centos-release-7-1.1503.el7.centos.2.8.x86_64 （@anaconda）
 From       : /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
Is this ok [y/N]: y  # 只有在第一次安装才会出现这个项目“确定要安装数码签章”才能继续！
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Warning: RPMDB altered outside of yum.
  Updating   : pam-1.1.8-12.el7_1.1.x86_64                                          1/3
  Installing : pam-devel-1.1.8-12.el7_1.1.x86_64                                    2/3
  Cleanup    : pam-1.1.8-12.el7.x86_64                                              3/3
  Verifying  : pam-1.1.8-12.el7_1.1.x86_64                                          1/3
  Verifying  : pam-devel-1.1.8-12.el7_1.1.x86_64                                    2/3
  Verifying  : pam-1.1.8-12.el7.x86_64                                              3/3

Installed:
  pam-devel.x86_64 0:1.1.8-12.el7_1.1

Dependency Updated:
  pam.x86_64 0:1.1.8-12.el7_1.1

Complete!
```

有没有很高兴啊！你不必知道软件在哪里，你不必手动下载软件，你也不必拿出原版光盘出来 mount 之后查询再安装！全部不需要，只要有了 yum 这个家伙，你的安装、升级再也不是什么难事！ 而且还能主动的进行软件的属性相依处理流程，如上所示，一口气帮我们处理好了所有事情！ 是不是很过瘾啊！而且整个动作完全免费！够酷吧！

- 移除功能：yum [remove] 软件

那能不能用 yum 移除软件呢？将刚刚的软件移除看看，会出现啥状况啊？

```
[root@study ~]# yum remove pam-devel
Loaded plugins: fastestmirror, langpacks
Resolving Dependencies   &lt;==同样的，先解决属性相依的问题
--&gt; Running transaction check
---&gt; Package pam-devel.x86_64 0:1.1.8-12.el7_1.1 will be erased
--&gt; Finished Dependency Resolution

Dependencies Resolved

==========================================================================================
 Package              Arch             Version                    Repository          Size
==========================================================================================
Removing:
 pam-devel            x86_64           1.1.8-12.el7_1.1           @updates           528 k

Transaction Summary
==========================================================================================
Remove  1 Package      # 还好！没有相依属性的问题，仅移除一个软件！

Installed size: 528 k
Is this ok [y/N]: y
Downloading packages:
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Erasing    : pam-devel-1.1.8-12.el7_1.1.x86_64                                      1/1
  Verifying  : pam-devel-1.1.8-12.el7_1.1.x86_64                                      1/1

Removed:
  pam-devel.x86_64 0:1.1.8-12.el7_1.1

Complete!
```

连移除也这么简单！看来，似乎不需要 rpm 这个指令也能够快乐的安装所有的软件了！ 虽然是如此，但是 yum 毕竟是架构在 rpm 上面所发展起来的，所以，鸟哥认为你还是得需要了解 rpm 才行！不要学了 yum 之后就将 rpm 的功能忘记了呢！切记切记！

22.3.2 yum 的配置文件

虽然 yum 是你的主机能够连线上 Internet 就可以直接使用的，不过，由于 CentOS 的映射站台可能会选错， 举例来说，我们在台湾，但是 CentOS 的映射站台却选择到了大陆北京或者是日本去，有没有可能发生啊！ 有啊！鸟哥教学方面就常常发生这样的问题，要知道，我们连线到大陆或日本的速度是非常慢的呢！那怎办？ 当然就是手动的修改一下 yum 的配置文件就好啰！

在台湾，CentOS 的映射站台主要有高速网络中心与义守大学，鸟哥近来比较偏好高速网络中心， 似乎更新的速度比较快，而且连接台湾学术网络也非常快速哩！因此，鸟哥下面建议台湾的朋友使用高速网络中心的 ftp 主机资源来作为 yum 服务器来源喔！不过因为鸟哥也在崑大服务，崑大目前也加入了 CentOS 的映射站， 如果在昆山或台南地区，也能够选择崑大的 FTP 喔！目前高速网络中心与崑大对于 CentOS 所提供的相关网址如下：

- http://ftp.twaren.net/Linux/CentOS/7/
- http://ftp.ksu.edu.tw/FTP/CentOS/7/

如果你连接到上述的网址后，就会发现里面有一堆链接，那些链接就是这个 yum 服务器所提供的软件库了！ 所以高速网络中心也提供了 centosplus, cloud, extras, fasttrack, os, updates 等软件库，最好认的软件库就是 os （系统默认的软件） 与 updates （软件升级版本） 啰！ 由于鸟哥在我的测试用主机是利用 x86_64 的版本， 因此那个 os 再点进去就会得到如下的可提供安装的网址：

- http://ftp.ksu.edu.tw/FTP/CentOS/7/os/x86_64/

为什么在上述的网址内呢？有什么特色！最重要的特色就是那个“ repodata ”的目录！该目录就是分析 RPM 软件后所产生的软件属性相依数据放置处！因此，当你要找软件库所在网址时， 最重要的就是该网址下面一定要有个名为 repodata 的目录存在！那就是软件库的网址了！ 其他的软件库正确网址，就请各位看倌自行寻找一下喔！现在让我们修改配置文件吧！

```
[root@study ~]# vim /etc/yum.repos.d/CentOS-Base.repo
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os&infra
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
```

如上所示，鸟哥仅列出 base 这个软件库内容而已，其他的软件库内容请自行查阅啰！上面的数据需要注意的是：

- [base]：代表软件库的名字！中括号一定要存在，里面的名称则可以随意取。但是不能有两个相同的软件库名称， 否则 yum 会不晓得该到哪里去找软件库相关软件清单文件。
- name：只是说明一下这个软件库的意义而已，重要性不高！
- mirrorlist=：列出这个软件库可以使用的映射站台，如果不想使用，可以注解到这行；
- baseurl=：这个最重要，因为后面接的就是软件库的实际网址！ mirrorlist 是由 yum 程序自行去捉映射站台， baseurl 则是指定固定的一个软件库网址！我们刚刚找到的网址放到这里来啦！
- enable=1：就是让这个软件库被启动。如果不想启动可以使用 enable=0 喔！
- gpgcheck=1：还记得 RPM 的数码签章吗？这就是指定是否需要查阅 RPM 文件内的数码签章！
- gpgkey=：就是数码签章的公钥档所在位置！使用默认值即可

了解这个配置文件之后，接下来让我们修改整个文件的内容，让我们这部主机可以直接使用高速网络中心的资源吧！ 修改的方式鸟哥仅列出 base 这个软件库项目而已，其他的项目请您自行依照上述的作法来处理即可！

```
[root@study ~]# vim /etc/yum.repos.d/CentOS-Base.repo
[base]
name=CentOS-$releasever - Base
baseurl=http://ftp.ksu.edu.tw/FTP/CentOS/7/os/x86_64/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

[updates]
name=CentOS-$releasever - Updates
baseurl=http://ftp.ksu.edu.tw/FTP/CentOS/7/updates/x86_64/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7

[extras]
name=CentOS-$releasever - Extras
baseurl=http://ftp.ksu.edu.tw/FTP/CentOS/7/extras/x86_64/
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-7
# 默认情况下，软件仓库仅有这三个有启用！所以鸟哥仅修改这三个软件库的 baseurl 而已喔！
```

接下来当然就是给它测试一下这些软件库是否正常的运行中啊！如何测试呢？再次使用 yum 即可啊！

```
范例一：列出目前 yum server 所使用的软件库有哪些？
[root@study ~]# yum repolist all
repo id                             repo name                          status
C7.0.1406-base/x86_64               CentOS-7.0.1406 - Base             disabled
C7.0.1406-centosplus/x86_64         CentOS-7.0.1406 - CentOSPlus       disabled
C7.0.1406-extras/x86_64             CentOS-7.0.1406 - Extras           disabled
C7.0.1406-fasttrack/x86_64          CentOS-7.0.1406 - CentOSPlus       disabled
C7.0.1406-updates/x86_64            CentOS-7.0.1406 - Updates          disabled
base                                CentOS-7 - Base                    enabled: 8,652
base-debuginfo/x86_64               CentOS-7 - Debuginfo               disabled
base-source/7                       CentOS-7 - Base Sources            disabled
centosplus/7/x86_64                 CentOS-7 - Plus                    disabled
centosplus-source/7                 CentOS-7 - Plus Sources            disabled
cr/7/x86_64                         CentOS-7 - cr                      disabled
extras                              CentOS-7 - Extras                  enabled:   181
extras-source/7                     CentOS-7 - Extras Sources          disabled
fasttrack/7/x86_64                  CentOS-7 - fasttrack               disabled
updates                             CentOS-7 - Updates                 enabled: 1,302
updates-source/7                    CentOS-7 - Updates Sources         disabled
repolist: 10,135
# 上面最右边有写 enabled 才是有启动的！由于 /etc/yum.repos.d/
# 有多个配置文件，所以你会发现还有其他的软件库存在。
```

- 修改软件库产生的问题与解决之道

由于我们是修改系统默认的配置文件，事实上，我们应该要在 /etc/yum.repos.d/ 下面新建一个文件， 该扩展名必须是 .repo 才行！但因为我们使用的是指定特定的映射站台，而不是其他软件开发商提供的软件库， 因此才修改系统默认配置文件。但是可能由于使用的软件库版本有新旧之分，你得要知道， yum 会先下载软件库的清单到本机的 /var/cache/yum 里面去！那我们修改了网址却没有修改软件库名称 （中括号内的文字）， 可能就会造成本机的清单与 yum 服务器的清单不同步，此时就会出现无法更新的问题了！

那怎么办啊？很简单，就清除掉本机上面的旧数据即可！需要手动处理吗？不需要的， 通过 yum 的 clean 项目来处理即可！

```
[root@study ~]# yum clean [packages&#124;headers&#124;all]
选项与参数：
 packages：将已下载的软件文件删除
 headers ：将下载的软件文件开始删除
 all     ：将所有软件库数据都删除！

范例一：删除已下载过的所有软件库的相关数据 （含软件本身与清单）
[root@study ~]# yum clean all
```

22.3.3 yum 的软件群组功能

通过 yum 来线上安装一个软件是非常的简单，但是，如果要安装的是一个大型专案呢？ 举例来说，鸟哥使用默认安装的方式安装了测试机，这部主机就只有 GNOME 这个窗口管理员，那我如果想要安装 KDE 呢？难道需要重新安装？当然不需要，通过 yum的软件群组功能即可！ 来看看指令先：

```
[root@study ~]# yum [群组功能] [软件群组]
选项与参数：
   grouplist   ：列出所有可使用的“软件群组组”，例如 Development Tools 之类的；
   groupinfo   ：后面接 group_name，则可了解该 group 内含的所有软件名；
   groupinstall：这个好用！可以安装一整组的软件群组，相当的不错用！
   groupremove ：移除某个软件群组；

范例一：查阅目前软件库与本机上面的可用与安装过的软件群组有哪些？
[root@study ~]# yum grouplist
Installed environment groups:            # 已经安装的系统环境软件群组
   Development and Creative Workstation
Available environment groups:            # 还可以安装的系统环境软件群组
   Minimal Install
   Compute Node
   Infrastructure Server
   File and Print Server
   Basic Web Server
   Virtualization Host
   Server with GUI
   GNOME Desktop
   KDE Plasma Workspaces
Installed groups:                        # 已经安装的软件群组！
   Development Tools
Available Groups:                        # 还能额外安装的软件群组！
   Compatibility Libraries
   Console Internet Tools
   Graphical Administration Tools
   Legacy UNIX Compatibility
   Scientific Support
   Security Tools
   Smart Card Support
   System Administration Tools
   System Management
Done
```

你会发现系统上面的软件大多是群组的方式一口气来提供安装的！还记全新安装 CentOS 时， 不是可以选择所需要的软件吗？而那些软件不是利用 GNOME/KDE/X Window ... 之类的名称存在吗？ 其实那就是软件群组啰！如果你执行上述的指令后，在“Available Groups”下面应该会看到一个 “Scientific Support”的软件群组，想知道那是啥吗？就这样做：

```
[root@study ~]# yum groupinfo "Scientific Support"
Group: Scientific Support
 Group-Id: scientific
 Description: Tools for mathematical and scientific computations, and parallel computing.
 Optional Packages:
   atlas
   fftw
   fftw-devel
   fftw-static
   gnuplot
   gsl-devel
   lapack
   mpich
....（以下省略）....
```

你会发现那就是一个科学运算、平行运算会用到的各种工具就是了！而下方则列出许多应该会在该群组安装时被下载与安装的软件们！ 让我们直接来安装看看！

```
[root@study ~]# yum groupinstall "Scientific Support"
```

正常情况下系统是会帮你安装好各项软件的。只是伤脑筋的是，刚刚好 Scientific Support 里面的软件都是“可选择的”！而不是“主要的 （mandatory）”， 因此默认情况下，上面这些软件通通不会帮你安装！！如果你想要安装上述的软件，可以使用 yum install atlas fftw .. 一个一个写进去安装～ 如果想要让 groupinstall 默认安装好所有的 optional 软件呢？那就得要修改配置文件！更改选 groupinstall 选择的软件项目即可！如下所示：

```
[root@study ~]# vim /etc/yum.conf
.....（前面省略）.....
distroverpkg=centos-release   # 找到这一行，下面新增一行！
group_package_types=default, mandatory, optional
.....（下面省略）.....

[root@study ~]# yum groupinstall "Scientific Support"
```

你就会发现系统开始进行了一大堆软件的安装！那就是啦！这个 group 功能眞是非常的方便呢！这个功能请一定要记下来，对你未来安装软件是非常有帮助的喔！ ^_^

22.3.4 EPEL/ELRepo 外挂软件以及自订配置文件

鸟哥因为工作的关系，在 Linux 上面经常需要安装第三方协力软件，这包括 NetCDF 以及 MPICH 等等的软件。现在由于平行处理的函数库需求大增， 所以 MPICH 已经纳入默认的 CentOS 7 软件库中。但是 NetCDF 这个软件就没有包含在里头了～同时，Linux 上面还有个很棒的统计软件，这个软件名称为“ R ”！ 默认也是不在 CentOS 的软件库内～唉～那怎办？要使用前一章介绍的 Tarball 去编译与安装吗？这倒不需要～因为有很多我们好棒的网友提供预先编译版本了！

在 Fedora 基金会里面发展了一个外加软件计划 （Extra Packages for Enterprise Linux, EPEL），这个计划主要是针对 Red Hat Enterprise Linux 的版本来开发的， 刚刚好 CentOS 也是针对 RHEL 的版本来处理的嘛！所以也就能够支持该软件库的相关软件相依环境了。这个计划的主网站在下面网页：

- https://fedoraproject.org/wiki/EPEL

而我们的 CentOS 7 主要可以使用的软件仓库网址为：

- https://dl.fedoraproject.org/pub/epel/7/x86_64/

除了上述的 Fedora 计划所提供的额外软件库之外，其实社群里面也有朋友针对 CentOS 与 EPEL 的不足而提供的许多软件仓库喔！ 下面鸟哥是列出当初鸟哥为了要处理 PCI passthrough 虚拟化而使用到的 ELRepo 这个软件仓库，若有其他的需求，你就得要自己搜寻了！ 这个 ELRepo 软件仓库与提供给 CentOS 7.x 的网址如下：

- http://elrepo.org/tiki/tiki-index.php
- http://elrepo.org/linux/elrepo/el7/x86_64
- http://elrepo.org/linux/kernel/el7/x86_64

这个 ELRepo 的软件库跟其他软件库比较不同的地方在于这个软件库提供的数据大多是与核心、核心模块与虚拟化相关软件有关，例如 NVidia 的驱动程序也在里面咧！ 尤其提供了最新的核心 （取名为 kernel-ml 的软件名称，其实就是最新的 Linux 核心啊！），如果你的系统像鸟哥的某些发展服务器一样，那就有可能会使用到这个软件库喔！

好了！根据上面的说明，来玩一玩下面这个仿真案例看看：

问：我的系统上面想要通过上述的 CentOS 7 的 EPEL 计划来安装 netcdf 以及 R 这两套软件，该如何处理？答：

- 首先，你的系统应该要针对 epel 进行 yum 的配置文件处理，处理方式如下：

  |

  ```
  [root@study ~]# vim /etc/yum.repos.d/epel.repo
  [epel]
  name = epel packages
  baseurl = https://dl.fedoraproject.org/pub/epel/7/x86_64/
  gpgcheck = 0
  enabled = 0
  ```

  |

  鸟哥故意不要启动这个软件仓库，只是未来有需要的时候才进行安装，默认不要去找这个软件库！

- 接下来使用这个软件库来进行安装 netcdf 与 R 的行为喔！

  |

```
[root@study ~]# yum --enablerepo=epel install netcdf R
```

|

这样就可以安装起来了！未来你没有加上 --enablerepo=epel 时，这个 EPEL 的软件并不会更新喔！

- 使用本机的原版光盘

万一你的主机并没有网络，但是你却有很多软件安装的需求～假设你的系统也都还没有任何升级的动作过， 这个时候我能不能用本机的光盘来作为主要的软件来源呢？答案当然是可以啊！那要怎么做呢？ 很简单，将你的光盘挂载到某个目录，我们这里还是继续假设在 /mnt 好了，然后设置如下的 yum 配置文件：

```
[root@study ~]# vim /etc/yum.repos.d/cdrom.repo
[mycdrom]
name = mycdrom
baseurl = file:///mnt
gpgcheck = 0
enabled = 0

[root@study ~]# yum --enablerepo=mycdrom install software_name
```

这个设置功能在你没有网络但是却需要解决很多软件相依性的状况时，相当好用啊！

22.3.5 全系统自动升级

我们可以手动选择是否需要升级，那能不能让系统自动升级，让我们的系统随时保持在最新的状态呢？ 当然可以啊！通过“ yum -y update ”来自动升级，那个 -y 很重要，因为可以自动回答 yes 来开始下载与安装！ 然后再通过 crontab 的功能来处理即可！假设我每天在台湾时间 3:00am 网络带宽比较轻松的时候进行升级， 你可以这样做的：

```
[root@study ~]# echo '10 1 * * * root /usr/bin/yum -y --enablerepo=epel update' &gt; /etc
[root@study ~]# vim /etc/crontab
```

从此你的系统就会自动升级啦！很棒吧！此外，你还是得要分析登录文件与收集 root 的信件的， 因为如果升级的是核心软件 （kernel），那么你还是得要重新开机才会让安装的软件顺利运行的！ 所以还是得分析登录文件，若有新核心安装，就重新开机，否则就让系统自动维持在最新较安全的环境吧！ 真是轻松愉快的管理啊！

22.3.6 管理的抉择：RPM 还是 Tarball

这一直是个有趣的问题：“如果我要升级的话，或者是全新安装一个新的软件， 那么该选择 RPM 还是 Tarball 来安装呢？”，事实上考虑的因素很多，不过鸟哥通常是这样建议的：

1. 优先选择原厂的 RPM 功能：

   由于原厂释出的软件通常具有一段时间的维护期，举例来说， RHEL 与 CentOS 每一个版本至少提供五年以上的更新期限。这对于我们的系统安全性来说，实在是非常好的选项！ 何解？既然 yum 可以自动升级，加上原厂会持续维护软件更新，那么我们的系统就能够自己保持在软件最新的状态， 对于资安来说当然会比较好一些的！ 此外，由于 RPM 与 yum 具有容易安装/移除/升级等特点，且还提供查询与验证的功能，安装时更有数码签章的保护， 让你的软件管理变的更轻松自在！因此，当然首选就是利用 RPM 来处理啦！

2. 选择软件官网释出的 RPM 或者是提供的软件库网址：

   不过，原厂并不会包山包海，因此某些特殊软件你的原版厂商并不会提供的！举例来说 CentOS 就没有提供 NTFS 的相关模块。此时你可以自行到官网去查阅，看看有没有提供相对到你的系统的 RPM 文件， 如果有提供软件库网址，那就更好啦！可以修改 yum 配置文件来加入该软件库，就能够自动安装与升级该软件！ 你说方不方便啊！

3. 利用 Tarball 安装特殊软件：

   某些特殊用途的软件并不会特别帮你制作 RPM 文件的，此时建议你也不要妄想自行制作 SRPM 来转成 RPM 啦！ 因为你只有区区一部主机而已，若是你要管理相同的 100 部主机，那么将源代码转制作成 RPM 就有价值！ 单机版的特殊软件，例如学术网络常会用到的 MPICH/PVM 等平行运算函数库，这种软件建议使用 tarball 来安装即可， 不需要特别去搜寻 RPM 啰！

4. 用 Tarball 测试新版软件：

   某些时刻你可能需要使用到新版的某个软件，但是原版厂商仅提供旧版软件，举例来说，我们的 CentOS 主要是定位于企业版，因此很多软件的要求是“稳”而不是“新”，但你就是需要新软件啊！ 然后又担心新软件装好后产生问题，回不到旧软件，那就惨了！此时你可以用 tarball 安装新软件到 /usr/local 下面， 那么该软件就能够同时安装两个版本在系统上面了！而且大多数软件安装数种版本时还不会互相干扰的！ 嘿嘿！用来作为测试新软件是很不错的呦！只是你就得要知道你使用的指令是新版软件还是旧版软件了！

所以说，RPM 与 Tarball 各有其优缺点，不过，如果有 RPM 的话，那么优先权还是在于 RPM 安装上面，毕竟管理上比较便利，但是如果软件的架构差异性太大， 或者是无法解决相依属性的问题，那么与其花大把的时间与精力在解决属性相依的问题上，还不如直接以 tarball 来安装，轻松又惬意！

22.3.7 基础服务管理：以 Apache 为例

我们在 17 章谈到 systemd 的服务管理，那个时候仅使用 vsftpd 这个比较简单的服务来做个说明，那是因为还没有谈到 yum 这个东东的缘故。 现在，我们已经处理好了网络问题 （20 章的内容），这个 yum 也能够顺利的使用！那么有没有其他的服务可以拿来做个测试呢？有

的，我们就拿网站服务器来说明吧！

一般来说， WWW 网站服务器需要的有 WWW 服务器软件 + 网页程序语言 + 数据库系统 + 程序语言与数据库的链接软件等等，在 CentOS 上面， 我们需要的软件就有“ httpd + php + mariadb-server + php-mysql ”这些软件。不过我们默认仅要启用 httpd 而已，因此等一下虽然上面的软件都要安装， 不过仅有 httpd 默认要启动而已喔！

另外，在默认的情况下，你无须修改服务的配置文件，都通过系统默认值来处理你的服务即可！那么有个江湖口诀你可以将它背下来～ 让你在处理服务的时候就不会掉漆了～

1. 安装： yum install （你的软件）
2. 启动： systemctl start （你的软件）
3. 开机启动： systemctl enable （你的软件）
4. 防火墙： firewall-cmd --add-service="（你的服务）"; firewall-cmd --permanent --add-service="（你的服务）"
5. 测试： 用软件去查阅你的服务正常与否～

下面就让我们一步一步来实验吧！

```
# 0\. 先检查一下有哪些软件没有安装或已安装～这个不太需要进行～单纯是鸟哥比较龟毛要先查看看而已！
[root@study ~]# rpm -q httpd php mariadb-server php-mysql
httpd-2.4.6-31.el7.centos.1.x86_64        # 只有这个安装好了，下面三个都没装！
package php is not installed
package mariadb-server is not installed
package php-mysql is not installed

# 1\. 安装所需要的软件！
[root@study ~]# yum install httpd php mariadb-server php-mysql
# 当然，大前提是你的网络没问题！这样就可以直接线上安装或升级！

# 2\. 3\. 启动与开机启动，这两个步骤要记得一定得进行！
[root@study ~]# systemctl daemon-reload
[root@study ~]# systemctl start httpd
[root@study ~]# systemctl enable httpd
[root@study ~]# systemctl status httpd
httpd.service - The Apache HTTP Server
   Loaded: loaded （/usr/lib/systemd/system/httpd.service; enabled）
   Active: active （running） since Wed 2015-09-09 16:52:04 CST; 9s ago
 Main PID: 8837 （httpd）
   Status: "Total requests: 0; Current requests/sec: 0; Current traffic:   0 B/sec"
   CGroup: /system.slice/httpd.service
           ├─8837 /usr/sbin/httpd -DFOREGROUND

# 4\. 防火墙
[root@study ~]# firewall-cmd --add-service="http"
[root@study ~]# firewall-cmd --permanent  --add-service="http"
[root@study ~]# firewall-cmd --list-all
public （default, active）
  interfaces: eth0
  sources:
  services: dhcpv6-client ftp http https ssh   # 这个是否有启动才是重点！
  ports: 222/tcp 555/tcp
  masquerade: no
  forward-ports:
  icmp-blocks:
  rich rules:
        rule family="ipv4" source address="192.168.1.0/24" accept
```

在最后的测试中，进入图形界面，打开你的浏览器，在网址列输入“ http://localhost ”就会出现如下的画面！ 那就代表成功了！你的 Linux 已经是 Web server 啰！就是这么简单！



图22.3.1、服务创建的第五步骤，测试一下有没有成功！

22.4 SRPM 的使用 ： rpmbuild （Optional）

谈完了 RPM 类型的软件之后，再来我们谈一谈包含了 Source code 的 SRPM 该如何使用呢？假如今天我们由网络上面下载了一个 SRPM 的文件，该如何安装他？又，如果我想要修改这个 SRPM 里面源代码的相关设置值，又该如何订正与重新编译呢？ 此外，最需要注意的是，新版的 rpm 已经将 RPM 与 SRPM 的指令分开了，SRPM 使用的是 rpmbuild 这个指令，而不是 rpm 喔！

22.4.1 利用默认值安装 SRPM 文件 （--rebuid/--recompile）

假设我下载了一个 SRPM 的文件，又不想要修订这个文件内的源代码与相关的设置值， 那么我可以直接编译并安装吗？当然可以！利用 rpmbuild 配合选项即可。选项主要有下面两个：

--rebuild	这个选项会将后面的 SRPM 进行“编译”与“打包”的动作，最后会产生 RPM 的文件，但是产生的 RPM 文件并没有安装到系统上。当你使用 --rebuild 的时候，最后通常会发现一行字体： `Wrote: /root/rpmbuild/RPMS/x86_64/pkgname.x86_64.rpm` 这个就是编译完成的 RPM 文件啰！这个文件就可以用来安装啦！安装的时候请加绝对路径来安装即可！
--recompile	这个动作会直接的“编译”“打包”并且“安装”啰！请注意， rebuild 仅“编译并打包”而已，而 recompile 不但进行编译跟打包，还同时进行“安装”了！

不过，要注意的是，这两个选项都没有修改过 SRPM 内的设置值，仅是通过再次编译来产生 RPM 可安装软件文件而已。 一般来说，如果编译的动作顺利的话，那么编译过程所产生的中间暂存盘都会被自动删除，如果发生任何错误， 则该中间文件会被保留在系统上，等待使用者的除错动作！

问：请由 http://vault.centos.org/ 下载正确的 CentOS 版本中， 在 updates 软件库当中的 ntp 软件 SRPM，请下载最新的那个版本即可，然后进行编译的行为。答：目前 （2015/09） 最新的版本为：ntp-4.2.6p5-19.el7.centos.1.src.rpm 这一个，所以我是这样作的：

- 先下载软件： wget http://vault.centos.org/7.1.1503/updates/Source/SPackages/ntp-4.2.6p5-19.el7.centos.1.src.rpm
- 再尝试直接编译看看： rpmbuild --rebuild ntp-4.2.6p5-19.el7.centos.1.src.rpm
- 上面的动作会告诉我还有一堆相依软件没有安装～所以我得要安装起来才行： yum install libcap-devel openssl-devel libedit-devel pps-tools-devel autogen autogen-libopts-devel
- 再次尝试编译的行为： rpmbuild --rebuild ntp-4.2.6p5-19.el7.centos.1.src.rpm
- 最终的软件就会被放置到： /root/rpmbuild/RPMS/x86_64/ntp-4.2.6p5-19.el7.centos.1.x86_64.rpm

上面的测试案例是将一个 SRPM 文件抓下来之后，依据你的系统重新进行编译。一般来说，因为该编译可能会依据你的系统硬件而最优化， 所以可能性能会好一些些，但是...人类根本感受不到那种性能优化的效果～所以并不建议你这么作。此外， 这种情况也很能发生在你从不同的 Linux distribution 所下载的 SRPM 拿来想要安装在你的系统上，这样作才算是有点意义。

一般来说，如果你有需要用到 SRPM 的文件，大部分的原因就是...你需要重新修改里面的某些设置，让软件加入某些特殊功能等等的。 所以啰，此时就得要将 SRPM 拆开，编辑一下编译配置文件，然后再予以重新编译啦！下个小节我们来玩玩修改设置的方式！

22.4.2 SRPM 使用的路径与需要的软件

SRPM 既然含有 source code ，那么其中必定有配置文件啰，所以首先我们必需要知道，这个 SRPM 在进行编译的时候会使用到哪些目录呢？这样一来才能够来修改嘛！ 不过从 CentOS 6.x 开始 （当然包含我们的 CentOS 7.x 啰），因为每个用户应该都有能力自己安装自己的软件，因此 SRPM 安装、设置、编译、最终结果所使用的目录都与操作者的主文件夹有关～鸟哥假设你用 root 的身份来进行 SRPM 的操作， 那么你应该就会使用到下列的目录喔：

/root/rpmbuild/SPECS	这个目录当中放置的是该软件的配置文件，例如这个软件的信息参数、设置项目等等都放置在这里；
/root/rpmbuild/SOURCES	这个目录当中放置的是该软件的原始文件 （*.tar.gz 的文件） 以及 config 这个配置文件；
/root/rpmbuild/BUILD	在编译的过程中，有些暂存的数据都会放置在这个目录当中；
/root/rpmbuild/RPMS	经过编译之后，并且顺利的编译成功之后，将打包完成的文件放置在这个目录当中。里头有包含了 x86_64, noarch.... 等等的次目录。
/root/rpmbuild/SRPMS	与 RPMS 内相似的，这里放置的就是 SRPM 封装的文件啰！有时候你想要将你的软件用 SRPM 的方式释出时， 你的 SRPM 文件就会放置在这个目录中了。



Tips 早期要使用 SRPM 时，必须是 root 的身份才能够使用编译行为，同时源代码都会被放置到 /usr/src/redhat/ 目录内喔！ 跟目前放置到 /~username/rpmbuild/ 的情况不太一样！

此外，在编译的过程当中，可能会发生不明的错误，或者是设置的错误，这个时候就会在 /tmp 下面产生一个相对应的错误文件，你可以根据该错误文件进行除错的工作呢！ 等到所有的问题都解决之后，也编译成功了，那么刚刚解压缩之后的文件，就是在 /root/rpmbild/{SPECS, SOURCES, BUILD} 等等的文件都会被杀掉，而只剩下放置在 /root/rpmbuild/RPMS 下面的文件了！

由于 SRPM 需要重新编译，而编译的过程当中，我们至少需要有 make 与其相关的程序，及 gcc, c, c++ 等其他的编译用的程序语言来进行编译，更多说明请参考第二十一章源代码所需基础软件吧。 所以，如果你在安装的过程当中没有选取软件开发工具之类的软件，这时就得要使用上一小节介绍的 yum 来安装就是了！ 当然，那个 "Development Tools" 的软件群组请不要忘记安装了！

问：尝试将上个练习下载的 ntp 的 SRPM 软件直接安装到系统中 （不要编译），然后查阅一下所有用到的目录为何？答：

```
# 1\. 鸟哥这里假设你用 root 的身份来进行安装的行为喔！
[root@study ~]# rpm -ivh ntp-4.2.6p5-19.el7.centos.1.src.rpm
Updating / installing...
   1:ntp-4.2.6p5-19.el7.centos.1      ################################# [100%]
warning: user mockbuild does not exist - using root
warning: group mockbuild does not exist - using root
# 会有一堆 warning 的问题，那个不要理它！可以忽略没问题的！

# 2\. 查阅一下 /root/rpmbuild 目录的内容！
[root@study ~]# ll -l /root/rpmbuild
drwxr-xr-x. 3 root root   39 Sep  8 16:16 BUILD
drwxr-xr-x. 2 root root    6 Sep  8 16:16 BUILDROOT
drwxr-xr-x. 4 root root   32 Sep  8 16:16 RPMS
drwxr-xr-x. 2 root root 4096 Sep  9 09:43 SOURCES
drwxr-xr-x. 2 root root   39 Sep  9 09:43 SPECS     # 这个家伙最重要！
drwxr-xr-x. 2 root root    6 Sep  8 14:51 SRPMS

[root@study ~]# ll -l /root/rpmbuild/{SOURCES,SPECS}
/root/rpmbuild/SOURCES:
-rw-rw-r--. 1 root root      559 Jun 24 07:44 ntp-4.2.4p7-getprecision.patch
-rw-rw-r--. 1 root root      661 Jun 24 07:44 ntp-4.2.6p1-cmsgalign.patch
.....（中间省略）.....
/root/rpmbuild/SPECS:
-rw-rw-r--. 1 root root   41422 Jun 24 07:44 ntp.spec   # 这就是重点！
```

22.4.3 配置文件的主要内容 （*.spec）

如前一个小节的练习，我们知道在 /root/rpmbuild/SOURCES 里面会放置原始文件 （tarball） 以及相关的修补档 （patch file）， 而我们也知道编译需要的步骤大抵就是 ./configure, make, make check, make install 等，那这些动作写入在哪里呢？ 就在 SPECS 目录中啦！让我们来瞧一瞧 SPECS 里面的文件说些什么吧！

```
[root@study ~]# cd /root/rpmbuild/SPECS
[root@study SPECS]# vim ntp.spec
# 1\. 首先，这个部分在介绍整个软件的基本相关信息！不论是版本还是释出次数等。
Summary: The NTP daemon and utilities            # 简易的说明这个软件的功能
Name: ntp                                        # 软件的名称
Version: 4.2.6p5                                 # 软件的版本
Release: 19%{?dist}.1                            # 软件的释出版次
# primary license （COPYRIGHT） : MIT             # 下面有很多 # 的注解说明！
```

```
.....（中间省略）.....
License: （MIT and BSD and BSD with advertising） and GPLv2
Group: System Environment/Daemons
Source0: http://www.eecis.udel.edu/~ntp/ntp_spool/ntp4/ntp-4.2/ntp-%{version}.tar.gz
Source1: ntp.conf                                # 写 SourceN 的就是源代码！
Source2: ntp.keys                                # 源代码可以有很多个！
.....（中间省略）.....
Patch1: ntp-4.2.6p1-sleep.patch                  # 接下来则是补丁文件，就是 PatchN 的目的！
Patch2: ntp-4.2.6p4-droproot.patch
.....（中间省略）.....

# 2\. 这部分则是在设置相依属性需求的地方！
URL: http://www.ntp.org                          # 下面则是说明这个软件的相依性，
Requires（post）: systemd-units                    # 还有编译过程需要的软件有哪些等等！
Requires（preun）: systemd-units
Requires（postun）: systemd-units
Requires: ntpdate = %{version}-%{release}
BuildRequires: libcap-devel openssl-devel libedit-devel perl-HTML-Parser
BuildRequires: pps-tools-devel autogen autogen-libopts-devel systemd-units
.....（中间省略）.....
%package -n ntpdate                              # 其实这个软件包含有很多次软件喔！
Summary: Utility to set the date and time via NTP
Group: Applications/System
Requires（pre）: shadow-utils
Requires（post）: systemd-units
Requires（preun）: systemd-units
Requires（postun）: systemd-units
.....（中间省略）.....

# 3\. 编译前的预处理，以及编译过程当中所需要进行的指令，都写在这里
#    尤其 %build 下面的数据，几乎就是 makefile 里面的信息啊！
%prep                                            # 这部份大多在处理补丁的动作！
%setup -q -a 5
%patch1 -p1 -b .sleep                            # 这些 patch 当然与前面的 PatchN 有关！
%patch2 -p1 -b .droproot
.....（中间省略）.....
%build                                           # 其实就是 ./configure, make 等动作！
sed -i 's&#124;$CFLAGS -Wstrict-overflow&#124;$CFLAGS&#124;' configure sntp/configure
export CFLAGS="$RPM_OPT_FLAGS -fPIE -fno-strict-aliasing -fno-strict-overflow"
export LDFLAGS="-pie -Wl,-z,relro,-z,now"
%configure \                                     # 不就是 ./configure 的意思吗！
        --sysconfdir=%{_sysconfdir}/ntp/crypto \
        --with-openssl-libdir=%{_libdir} \
        --without-ntpsnmpd \
        --enable-all-clocks --enable-parse-clocks \
        --enable-ntp-signd=%{_localstatedir}/run/ntp_signd \
        --disable-local-libopts
echo '#define KEYFILE "%{_sysconfdir}/ntp/keys"' &gt;&gt; ntpdate/ntpdate.h
echo '#define NTP_VAR "%{_localstatedir}/log/ntpstats/"' &gt;&gt; config.h

make %{?_smp_mflags}                             # 不就是 make 了吗！
.....（中间省略）.....

%install                                         # 就是安装过程所进行的各项动作了！
make DESTDIR=$RPM_BUILD_ROOT bindir=%{_sbindir} install

mkdir -p $RPM_BUILD_ROOT%{_mandir}/man{5,8}
sed -i 's/sntp\.1/sntp\.8/' $RPM_BUILD_ROOT%{_mandir}/man1/sntp.1
mv $RPM_BUILD_ROOT%{_mandir}/man{1/sntp.1,8/sntp.8}
rm -rf $RPM_BUILD_ROOT%{_mandir}/man1
.....（中间省略）.....

# 4\. 这里列出，这个软件释出的文件有哪些的意思！
%files                                           # 这软件所属的文件有哪些的意思！
%dir %{ntpdocdir}
%{ntpdocdir}/COPYRIGHT
%{ntpdocdir}/ChangeLog
.....（中间省略）.....

# 5\. 列出这个软件的更改历史纪录档！
%changelog
* Tue Jun 23 2015 CentOS Sources &lt;bugs@centos.org&gt; - 4.2.6p5-19.el7.centos.1
```

```
- rebrand vendorzone

* Thu Apr 23 2015 Miroslav Lichvar &lt;mlichvar@redhat.com&gt; 4.2.6p5-19.el7_1.1
- don't step clock for leap second with -x option （#1191122）
.....（后面省略）.....
```

要注意到的是 ntp.sepc 这个文件，这是主要的将 SRPM 编译成 RPM 的配置文件，他的基本规则可以这样看：

1. 整个文件的开头以Summary为开始，这部份的设置都是最基础的说明内容；
2. 然后每个不同的段落之间，都以 % 来做为开头，例如 %prep 与 %install 等；

我们来谈一谈几个常见的 SRPM 设置段落：

- 系统整体信息方面：

刚刚你看到的就有下面这些重要的咚咚啰：

参数	参数意义
Summary	本软件的主要说明，例如上表中说明了本软件是针对 NTP 的软件功能与工具等啦！
Name	本软件的软件名称 （最终会是 RPM 文件的文件名构成之一）
Version	本软件的版本 （也会是 RPM 文件名的构成之一）
Release	这个是该版本打包的次数说明 （也会是 RPM 文件名的构成之一）。由于我们想要动点手脚，所以请将“ 19%{?dist}.1 ” 修改为“ 20.vbird ” 看看
License	这个软件的授权模式，看起来涵盖了所有知名的 Open source 授权啊！！
Group	这个软件在安装的时候，主要是放置于哪一个软件群组当中 （yum grouplist 的特点！）；
URL	这个源代码的主要官方网站；
SourceN	这个软件的来源，如果是网络上下载的软件，通常一定会有这个信息来告诉大家这个原始文件的来源！ 此外，如果有多个软件来源，就会以 Source0, Source1... 来处理源代码喔！
PatchN	就是作为补丁的 patch file 啰！也是可以有好多个！
BuildRoot	设置作为编译时，该使用哪个目录来暂存中间文件 （如编译过程的目标文件/链接文件等档）。
上述为必须要存在的项目，下面为可使用的额外设置值	
Requires	如果你这个软件还需要其他的软件的支持，那么这里就必需写上来，则当你制作成 RPM 之后，系统就会自动的去检查啦！这就是“相依属性”的主要来源啰！
BuildRequires	编译过程中所需要的软件。Requires 指的是“安装时需要检查”的，因为与实际运行有关，这个 BuildRequires 指的是“编译时”所需要的软件，只有在 SRPM 编译成为 RPM 时才会检查的项目。

上面几个数据通常都必需要写啦！但是如果你的软件没有相依属性的关系时，那么就可以不需要那个 Requires 啰！ 根据上面的设置，最终的文件名就会是“{Name}-{Version}-{Release}.{Arch}.rpm”的样式， 以我们上面的设置来说，文件名应该会是“ntp-4.2.6p5-20.vbird.x86_64.rpm”的样子啰！

- %description：

将你的软件做一个简短的说明！这个也是必需要的。还记得使用“ rpm -qi 软件名称 ”会出现一些基础的说明吗？ 上面这些东西包括 Description 就是在显示这些重要信息的啦！所以，这里记得要详加解释喔！

- %prep：

pre 这个关键字原本就有“在...之前”的意思，因此这个项目在这里指的就是“尚未进行设置或安装之前，你要编译完成的 RPM 帮你事先做的事情”，就是 prepare 的简写啰！那么他的工作事项主要有：

1. 进行软件的补丁 （patch） 等相关工作；
2. 寻找软件所需要的目录是否已经存在？确认用的！
3. 事先创建你的软件所需要的目录，或者事先需要进行的任务；
4. 如果待安装的Linux系统内已经有安装的时候可能会被覆盖掉的文件时，那么就必需要进行备份（backup）的工作了！

在本案例中，你会发现程序会使用 patch 去进行补丁的动作啦！所以程序的源代码才会更新到最新啊！

- %build：

build 就是创建啊！所以当然啰，这个段落就是在谈怎么 make 编译成为可执行的程序啰！ 你会发现在此部分的程序码方面，就是 ./configure, make 等项目哩！一般来说，如果你会使用 SRPM 来进行重新编译的行为， 通常就是要重新 ./configure 并给予新的参数设置！于是这部份就可能会修改到！

- %install：

编译完成 （build） 之后，就是要安装啦！安装就是写在这里，也就是类似 Tarball 里面的 make install 的意思啰！

- %files：

这个软件安装的文件都需要写到这里来，当然包括了“目录”喔！所以连同目录请一起写到这个段落当中！以备查验呢！^_^ ！此外，你也可以指定每个文件的类型，包括文档文件 （%doc 后面接的） 与配置文件 （%config 后面接的） 等等。

- %changelog：

这个项目主要则是在记录这个软件曾经的更新纪录啰！星号 （*） 后面应该要以时间，修改者， email 与软件版本来作为说明， 减号 （-） 后面则是你要作的详细说明啰！在这部份鸟哥就新增了两行，内容如下：

```
%changelog
* Wed Sep 09 2015 VBird Tsai &lt;vbird@mail.vbird.idv.tw&gt;- 4.2.6p5-20.vbird
- only rbuild this SRPM to RPM

* Tue Jun 23 2015 CentOS Sources &lt;bugs@centos.org&gt; - 4.2.6p5-19.el7.centos.1
- rebrand vendorzone
....（下面省略）....
```

修改到这里也差不多了，您也应该要了解到这个 ntp.spec 有多么重要！我们用 rpm -q 去查询一堆信息时， 其实都是在这里写入的！这样了解否？接下来，就让我们来了解一下如何将 SRPM 给他编译出 RPM 来吧！

22.4.4 SRPM 的编译指令 （-ba/-bb）

要将在 /root/rpmbuild 下面的数据编译或者是单纯的打包成为 RPM 或 SRPM 时，就需要 rpmbuild 指令与相关选项的帮忙了！我们只介绍两个常用的选项给您了解一下：

```
[root@study ~]# rpmbuild -ba ntp.spec  &lt;==编译并同时产生 RPM 与 SRPM 文件
[root@study ~]# rpmbuild -bb ntp.spec  &lt;==仅编译成 RPM 文件
```

这个时候系统就会这样做：

1. 先进入到 BUILD 这个目录中，亦即是： /root/rpmbuild/BUILD 这个目录；
2. 依照 *.spec 文件内的 Name 与 Version 定义出工作的目录名称，以我们上面的例子为例，那么系统就会在 BUILD 目录中先删除 ntp-4.2.6p5 的目录，再重新创建一个 ntp-4.2.6p5 的目录，并进入该目录；
3. 在新建的目录里面，针对 SOURCES 目录下的来源文件，也就是 *.spec* 里面的 *Source* 设置的那个文件，以 *tar* 进行解压缩，以我们这个例子来说，则会在 */root/rpmbuild/BUILD/ntp-4.2.6p5* 当中，将 */root/rpmbuild/SOURCES/ntp-* 等等多个源代码文件进行解压缩啦！
4. 再来开始 %build 及 %install 的设置与编译！
5. 最后将完成打包的文件给他放置到该放置的地方去，如果你的系统是 x86_64 的话，那么最后编译成功的 *.x86_64.rpm文件就会被放置在 /root/rpmbuild/RPMS/x86_64 里面啰！如果是 noarch 那么自然就是 /root/rpmbuild/RPMS/noarch 目录下啰！

整个步骤大概就是这样子！最后的结果数据会放置在 RPMS 那个目录下面就对啦！我们这个案例中想要同时打包 RPM 与 SRPM ， 因此请您自行处理一下“ rpmbuild -ba ntp.spec ”吧！

```
[root@study ~]# cd /root/rpmbuild/SPECS
[root@study SPECS]# rpmbuild -ba ntp.spec
.....（前面省略）.....
Wrote: /root/rpmbuild/SRPMS/ntp-4.2.6p5-20.vbird.src.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/ntp-4.2.6p5-20.vbird.x86_64.rpm
Wrote: /root/rpmbuild/RPMS/noarch/ntp-perl-4.2.6p5-20.vbird.noarch.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/ntpdate-4.2.6p5-20.vbird.x86_64.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/sntp-4.2.6p5-20.vbird.x86_64.rpm
Wrote: /root/rpmbuild/RPMS/noarch/ntp-doc-4.2.6p5-20.vbird.noarch.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/ntp-debuginfo-4.2.6p5-20.vbird.x86_64.rpm
Executing（%clean）: /bin/sh -e /var/tmp/rpm-tmp.xZh6yz
+ umask 022
+ cd /root/rpmbuild/BUILD
+ cd ntp-4.2.6p5
+ /usr/bin/rm -rf /root/rpmbuild/BUILDROOT/ntp-4.2.6p5-20.vbird.x86_64
+ exit 0

[root@study SPECS]# find /root/rpmbuild -name 'ntp*rpm'
/root/rpmbuild/RPMS/x86_64/ntp-4.2.6p5-20.vbird.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/ntpdate-4.2.6p5-20.vbird.x86_64.rpm
/root/rpmbuild/RPMS/x86_64/ntp-debuginfo-4.2.6p5-20.vbird.x86_64.rpm
/root/rpmbuild/RPMS/noarch/ntp-perl-4.2.6p5-20.vbird.noarch.rpm
/root/rpmbuild/RPMS/noarch/ntp-doc-4.2.6p5-20.vbird.noarch.rpm
/root/rpmbuild/SRPMS/ntp-4.2.6p5-20.vbird.src.rpm
# 上面分别是 RPM 与 SRPM 的文件文件名！
```

您瞧！嘿嘿～有 vbird 的软件出现了！相当有趣吧！另外，有些文件软件是与硬件等级无关的（因为单纯的文件啊！），所以如上表所示， 你会发现 ntp-doc-4.2.6p5-20.vbird.noarch.rpm 是 noarch 喔！有趣吧！

22.4.5 一个打包自己软件的范例

这个就有趣了！我们自己来编辑一下自己制作的 RPM 怎么样？会很难吗？完全不会！ 我们这里就举个例子来玩玩吧！还记得我们在前一章谈到 Tarball 与 make 时，曾经谈到的 main 这个程序吗？现在我们将这个程序加上 Makefile 后， 将他制作成为 main-0.1-1.x86_64.rpm 好吗？那该如何进行呢？下面就让我们来处理处理吧！

- 制作源代码文件 tarball 产生：

因为鸟哥的网站并没有直接释出 main-0.2，所以假设官网提供的是 main-0.l 版本之外，同时提供了一个 patch 文件～ 那我们就得要这样作：

- main-0.1.tar.gz 放在 /root/rpmbuild/SOURCES/
- main_0.1_to_0.2_patch 放在 /root/rpmbuild/SOURCES/
- main.spec 自行撰写放在 /root/rpmbuild/SPECS/

```
# 1\. 先来处理源代码的部份，假设你的 /root/rpmbuild/SOURCES 已经存在了喔！
[root@study ~]# cd /root/rpmbuild/SOURCES
[root@study SOURCES]# wget http://linux.vbird.org/linux_basic/0520source/main-0.1.tgz
[root@study SOURCES]# wget http://linux.vbird.org/linux_basic/0520source/main_0.1_to_0.2.
[root@study SOURCES]# ll main*
-rw-r--r--. 1 root root  703 Sep  4 14:47 main-0.1.tgz
-rw-r--r--. 1 root root 1538 Sep  4 14:51 main_0.1_to_0.2.patch
```

接下来就是 spec 文件的创建啰！

- 创建 *.spec 的配置文件

这个文件的创建是所有 RPM 制作里面最重要的课题！你必须要仔细的设置他，不要随便处理！仔细看看吧！ 有趣的是，CentOS 7.x 会主动的将必要的设置参数列出来喔！相当有趣！^_^

```
[root@study ~]# cd /root/rpmbuild/SPECS
[root@study SPECS]# vim main.spec
Name:           main
Version:        0.1
Release:        1%{?dist}
Summary:        Shows sin and cos value.
Group:          Scientific Support
License:        GPLv2
URL:            http://linux.vbird.org/
Source0:        main-0.1.tgz             # 这两个文件名要正确喔！
Patch0:         main_0.1_to_0.2.patch

%description
This package will let you input your name and calculate sin cos value.

%prep
%setup -q
%patch0 -p1                              # 要用来作为 patch 的动作！

%build
make clean main                          # 编译就好！不要安装！

%install
mkdir -p %{buildroot}/usr/local/bin
install -m 755 main %{buildroot}/usr/local/bin # 这才是顺利的安装行为！

%files
/usr/local/bin/main

%changelog
* Wed Sep 09 2015 VBird Tsai &lt;vbird@mail.vbird.idv.tw&gt; 0.2
- build the program
```

- 编译成为 RPM 与 SRPM

老实说，那个 spec 文件创建妥当后，后续的动作就简单的要命了！开始来编译吧！

```
[root@study SPECS]# rpmbuild -ba main.spec
.....（前面省略）.....
Wrote: /root/rpmbuild/SRPMS/main-0.1-1.el7.centos.src.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/main-0.1-1.el7.centos.x86_64.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/main-debuginfo-0.1-1.el7.centos.x86_64.rpm
```

很快的，我们就已经创建了几个 RPM 文件啰！接下来让我们好好测试一下打包起来的成果吧！

- 安装/测试/实际查询

```
[root@study ~]# yum install /root/rpmbuild/RPMS/x86_64/main-0.1-1.el7.centos.x86_64.rpm
[root@study ~]# rpm -ql main
/usr/local/bin/main   &lt;==自己尝试执行 main 看看！

[root@study ~]# rpm -qi main
Name        : main
Version     : 0.1
Release     : 1.el7.centos
Architecture: x86_64
Install Date: Wed 09 Sep 2015 04:29:08 PM CST
Group       : Scientific Support
Size        : 7200
License     : GPLv2
Signature   : （none）
Source RPM  : main-0.1-1.el7.centos.src.rpm
Build Date  : Wed 09 Sep 2015 04:27:29 PM CST
Build Host  : study.centos.vbird
Relocations : （not relocatable）
URL         : http://linux.vbird.org/
Summary     : Shows sin and cos value.
Description :
This package will let you input your name and calculate sin cos value.
# 看到没？属于你自己的软件喔！真是很愉快的啦！
```

用很简单的方式，就可以将自己的软件或者程序给他修改与设置妥当！以后你就可以自行设置你的 RPM 啰！当然，也可以手动修改你的 SRPM 的来源文件内容啰！

22.5 重点回顾

- 为了避免使用者自行编译的困扰，开发商自行在特定的硬件与操作系统平台上面预先编译好软件， 并将软件以特殊格式封包成文件，提供终端用户直接安装到固定的操作系统上，并提供简单的查询/安装/移除等流程。 此称为软件管理员。常见的软件管理员有 RPM 与 DPKG 两大主流。
- RPM 的全名是 RedHat Package Manager，原本是由 Red Hat 公司所发展的，流传甚广；
- RPM 类型的软件中，所含有的软件是经过编译后的 binary program ，所以可以直接安装在使用者端的系统上， 不过，也由于如此，所以 RPM 对于安装者的环境要求相当严格；
- RPM 除了将软件安装至使用者的系统上之外，还会将该软件的版本、名称、文件与目录配置、系统需求等等均记录于数据库 （/var/lib/rpm） 当中，方便未来的查询与升级、移除；
- RPM 可针对不同的硬件等级来加以编译，制作出来的文件可于扩展名 （i386, i586, i686, x86_64, noarch） 来分辨；
- RPM 最大的问题为软件之间的相依性问题；
- SRPM 为 Source RPM ，内含的文件为 Source code 而非为 binary file ，所以安装 SRPM 时还需要经过 compile ，不过，SRPM 最大的优点就是可以让使用者自行修改设置参数 （makefile/configure 的参数） ，以符合使用者自己的 Linux 环境；
- RPM 软件的属性相依问题，已经可以借由 yum 或者是 APT 等方式加以克服。 CentOS 使用的就是 yum 机制。
- yum 服务器提供多个不同的软件库放置个别的软件，以提供用户端分别管理软件类别。

22.6 本章习题

- 情境仿真题：通过 EPEL 安装 NTFS 文件系统所需要的软件
    - 目标：利用 EPEL 提供的软件来搜寻是否有 NTFS 所需要的各项模块！；
    - 目标：你的 Linux 必须要已经接上 Internet 才行；
    - 需求：最好了解磁盘容量是否够用，以及如何启动服务等。 其实这个任务非常简单！因为我们在前面各小节的说明当中已经说明了如何设置 EPEL 的 yum 配置文件，此时你只要通过下面的方式来处理即可：
    - 使用 yum --enablerepo=epel search ntfs 找出所需要的软件名称
    - 再使用 yum --enablerepo=epel install ntfs-3g ntfsprogs 来安装即可！

简答题部分：

- 如果你曾经修改过 yum 配置文件内的软件库设置 （/etc/yum.repos.d/*.repo） ，导致下次使用 yum 进行安装时老是发现错误， 此时你该如何是好？先确认你的配置文件确实是正确的，如果没问题，可以将 yum 的高速缓存清除，使用“yum clean all”即可。 事实上， yum 的所有高速缓存、下载软件、下载软件的表头数据，都放置于 /var/cache/yum/ 目录下。
- 简单说明 RPM 与 SRPM 的异同？RPM 文件是由程序打包者 （通常是由 distribution 的开发商） 借由程序的源代码，在特定的平台上面所编译成功的 binary program 的数据，并将该数据制作成为 RPM 的格式，以方便相同软、硬件平台的使用者之安装使用。 在安装时显的很简单，因为程序打包者的平台与使用者所使用的平台默认为相同。 至于 SRPM 则是借由与 RPM 相同的配置文件数据，不过将源代码直接包在 SRPM 文件当中，而不经过编译。 因为 SRPM 所内含的数据为源代码，所以安装时必须要再经过编译的行为才能成为 RPM 并提供使用者安装。
- 假设我想要安装一个软件，例如 pkgname.i386.rpm ，但却老是发生无法安装的问题，请问我可以加入哪些参数来强制安装他？可以加入 --nodeps 等参数。例如 rpm -ivh --nodeps pkgname.i386.rpm
- 承上题，你认为强制安装之后，该软件是否可以正常执行？为什么？一般来说，应该是“不能执行”的，因为该软件具有相依属性的问题， 某些时刻该软件的程序可能需要调用外部的函数库，但函数库可能未安装，因此当然无法执行成功。
- 有些人使用 CentOS 7.x 安装在自己的 Atom CPU 上面，却发现无法安装，在查询了该原版光盘的内容，发现里面的文件名称为 *.x86_64.rpm 。请问，无法安装的可能原因为何？Atom 虽然也是属于 x86 的架构，但是某些 atom 是属于 32 位的系统。但是 CentOS 7 已经仅释出 64 位的版本，所以当然无法安装了！
- 请问我使用 rpm -Fvh *.rpm 及 rpm -Uvh* .rpm 来升级时，两者有何不同？-Uvh 后面接的

软件，如果原本未安装，则直接安装，原本已安装时，则直接升级；-Fvh 后面接的软件，如果原本未安装，则不安装，原本已安装时，则直接升级；

- 假设有一个厂商推出软件时，自行处理了数码签章，你想要安装他们的软件所以需要使用数码签章，假设数码签章的文件名为 signe， 那你该如何安装？rpm --import signe
- 承上，假设该软件厂商提供了 yum 的安装网址为： http://their.server.name/path/ ，那你该如何处理 yum 的配置文件？可以自行取个文件名，在此例中我们使用“ vim /etc/yum.repos.d/their.repo ”，扩展名要正确！ 内容有点像这样即可：

```
[their]
name=their server name
baseurl=http://their.server.name/path/
enable=1
gpgcheck=0
```

然后使用 yum 去安装该软件看看。

22.7 参考资料与延伸阅读

- [1]GNU Privacy Guard （GPG） 官方网站的介绍：http://www.gnupg.org/
- RPM 包装文件管理程序：http://www.study-area.org/tips/rpm.htm
- 中文 RPM HOW-TO：http://www.linux.org.tw/CLDP/RPM-HOWTO.html
- RPM 的使用：http://linux.tnc.edu.tw/techdoc/rpm-howto.htm
- 大家来作 RPM ：http://freebsd.ntu.edu.tw/bsd/4/3/2/29.html
- 一本 RPM 的原文书：http://linux.tnc.edu.tw/techdoc/maximum-rpm/rpmbook/
- 台湾网络危机处理小组：http://www.cert.org.tw/

2002/08/21：第一次完成 2003/02/11：重新编排与加入 FAQ 2004/04/11：已经完成了 Source code 与 Tarball ，开始进行 RPM 与 SRPM 的介绍！（需要耗时多日啊！因为又要进兵营去了！） 2004/04/20：终于给他熬出来啦！又是过了两个休假期间～啊！给我退伍令、其余免谈！ 2005/10/02：旧版的 SRPM 数据已经移动到 此处 。 2005/10/03：旧版的针对 Red Hat 与 Mandriva 的版本移动到 此处。 2005/10/03：将原本去年的版本改为 FC4 为范例的模样！ 2009/06/20：原本的针对 FC4 写的旧版文章移动到此处。 2009/09/18：加入了简单的情境仿真，也加入了一些关于 yum 的习题喔！ 2015/10/16：加入了 ELRepo 这个专门提供核心给 CentOS 使用的软件库功能介绍！

第二十三章、X Window 设置介绍

最近更新日期：20//

在 Linux 上头的图形接口我们称之为 X Window System，简称为 X 或 X11 啰！ 为何称之为系统呢？这是因为 X 窗口系统又分为 X server 与 X client ，既然是 Server/Client （主从架构） 这就表示其实 X 窗口系统是可以跨网络且跨平台的！X 窗口系统对于 Linux 来说仅是一个软件， 只是这个软件日趋重要喔！因为 Linux 是否能够在桌面电脑上面流行，与这个 X 窗口系统有关啦！ 好在，目前的 X 窗口系统整合到 Linux 已经非常优秀了，而且也能够具有 3D 加速的功能， 只是，我们还是得要了解一下 X 窗口系统才好，这样如果出问题，我们才有办法处理啊！

23.1 什么是 X Window System

Unix Like 操作系统不是只能进行服务器的架设而已，在美编、排版、制图、多媒体应用上也是有其需要的。 这些需求都需要用到图形接口 （Graphical User Interface, GUI） 的操作的， 所以后来才有所谓的 X Window System 这玩意儿。那么为啥图形窗口接口要称为 X 呢？因为就英文字母来看 X 是在 W（indow） 后面，因此，人们就戏称这一版的窗口接口为 X 啰 （有下一版的新窗口之意）！

事实上， X Window System 是个非常大的架构，他还用到网络功能呢！也就是说，其实 X 窗口系统是能够跨网络与跨操作系统平台的！ 而鸟哥这个基础篇是还没有谈到服务器与网络主从式架构，因此 X 在这里并不容易理解的。不过，没关系！ 我们还是谈谈 X 怎么来的，然后再来谈谈这 X 窗口系统的元件有哪些，慢慢来，应该还是能够理解 X 的啦！

23.1.1 X Window 的发展简史

X Window 系统最早是由 MIT （Massachusetts Institute of Technology, 麻省理工学院） 在 1984 年发展出来的， 当初 X 就是在 Unix 的 System V 这个操作系统版本上面开发出来的。在开发 X 时，开发者就希望这个窗口接口不要与硬件有强烈的相关性，这是因为如果与硬件的相关性高，那就等于是一个操作系统了， 如此一来的应用性会比较局限。因此 X 在当初就是以应用程序的概念来开发的，而非以操作系统来开发。

由于这个 X 希望能够通过网络进行图形接口的存取，因此发展出许多的 X 通讯协定，这些网络架构非常的有趣， 所以吸引了很多厂商加入研发，因此 X 的功能一直持续在加强！一直到 1987 年更改 X 版本到 X11 ，这一版 X 取得了明显的进步， 后来的窗口接口改良都是架构于此一版本，因此后来 X 窗口也被称为 X11 。这个版本持续在进步当中，到了 1994 年发布了新版的 X11R6 ，后来的架构都是沿用此一释出版本，所以后来的版本定义就变成了类似 1995 年的 X11R6.3 之类的样式。 [1]

1992 年 XFree86 （http://www.xfree86.org/） 计划顺利展开， 该计划持续在维护 X11R6 的功能性，包括对新硬件的支持以及更多新增的功能等等。当初定名为 XFree86 其实是根据“ X + Free software + x86 硬件 ”而来的呢。早期 Linux 所使用的 X Window 的主要核心都是由 XFree86 这个计划所提供的，因此，我们常常将 X 系统与 XFree86 挂上等号的说。

不过由于一些授权的问题导致 XFree86 无法继续提供类似 GPL 的自由软件，后来 Xorg 基金会就接手 X11R6 的维护！ Xorg （http://www.x.org/） 利用当初 MIT 发布的类似自由软件的授权， 将 X11R6 拿来进行维护，并且在 2004 年发布了 X11R6.8 版本，更在 2005 年后发表了 X11R7.x 版。 现在我们 CentOS 7.x 使用的 X 就是 Xorg 提供的 X11R7.X 喔！ 而这个 X11R6/X11R7 的版本是自由软件，因此很多组织都利用这个架构去设计他们的图形接口喔！包括 Mac OS X v10.3 也曾利用过这个架构来设计他们的窗口呢！我们的 CentOS 也是利用 Xorg 提供的 X11 啦！

从上面的说明，我们可以知道的是：

- 在 Unix Like 上面的图形使用者接口 （GUI） 被称为 X 或 X11；
- X11 是一个“软件”而不是一个操作系统；
- X11 是利用网络架构来进行图形接口的执行与绘制；
- 较著名的 X 版本为 X11R6 这一版，目前大部分的 X 都是这一版演化出来的 （包括 X11R7） ；
- 现在大部分的 distribution 使用的 X 都是由 Xorg 基金会所提供的 X11 软件；
- X11 使用的是 MIT 授权，为类似 GPL 的开放源代码授权方式。

23.1.2 主要元件： X Server/X Client/Window Manager/Display Manager

如同前面谈到的，X Window system 是个利用网络架构的图形使用者接口软件，那到底这个架构可以分成多少个元件呢？ 基本上是分成 X Server 与 X Client 两个元件而已喔！其中 X Server 在管理硬件，而 X Client 则是应用程序。 在运行上，X Client 应用程序会将所想要呈现的画面告知 X Server ，最终由 X server 来将结果通过他所管理的硬件绘制出来！ 整体的架构我们大约可以使用如下的图示来作个介绍：[2]

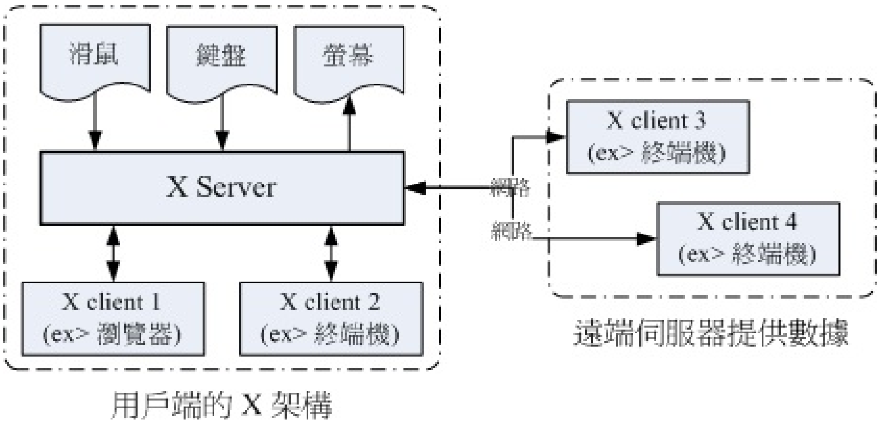


图23.1.1、X Window System 的架构

上面的图示非常有趣喔！我们在用户端想要取得来自服务器的图形数据时，我们用户端使用的当然是用户端的硬件设备啊， 所以，X Server 的重点就是在管理用户端的硬件，包括接受键盘/鼠标等设备的输入信息， 并且将图形绘制到屏幕上 （请注意上图的所有元件之间的箭头指示） 。但是到底要绘制个啥东西呢？ 绘图总是需要一些数据才能绘制吧？此时 X Client （就是 X 应用程序） 就很重要啦！他主要提供的就是告知 X Server 要绘制啥东西。那照这样的想法来思考，我们是想要取得远端服务器的绘图数据来我们的计算机上面显示嘛！ 所以啰，远端服务器提供的是 X client 软件啊！

下面就让我们来更深入的聊一聊这两个元件吧！

- X Server：硬件管理、屏幕绘制与提供字体功能：

既然 X Window System 是要显示图形接口，因此理所当然的需要一个元件来管理我主机上面的所有硬件设备才行！ 这个任务就是 X Server 所负责的。而我们在 X 发展简史当中提到的 XFree86 计划及 Xorg 基金会，主要提供的就是这个 X Server 啦！那么 X Server 管理的设备主要有哪些呢？其实与输入/输出有关喔！包括键盘、鼠标、手写板、显示器 （monitor） 、屏幕分辨率与色彩深度、显卡 （包含驱动程序） 与显示的字体等等，都是 X Server 管理的。

咦！显卡、屏幕以及键盘鼠标的设置，不是在开机的时候 Linux 系统以 systemd 的相关设置处理好了吗？ 为何 X Server 还要重新设置啊？这是因为 X Window 在 Linux 里面仅能算是“一套很棒的软件”， 所以 X Window 有自己的配置文件，你必须要针对他的配置文件设置妥当才行。也就是说， Linux 的设置与 X Server 的设置不一定要相同的！因此，你在 CentOS 7 的 multi-user.target 想要玩图形接口时，就得要载入 X Window 需要的驱动程序才行～总之， X Server 的主要功能就是在管理“主机”上面的显示硬件与驱动程序。

既然 X Window System 是以通过网络取得图形接口的一个架构，那么用户端是如何取得服务器端提供的图形画面呢？ 由于服务器与用户端的硬件不可能完全相同，因此我们用户端当然不可能使用到服务器端的硬件显示功能！ 举例来说，你的用户端计算机并没有 3D 影像加速功能，那么你的画面可能呈现出服务器端提供的 3D 加速吗？ 当然不可能吧！所以啰 X Server 的目的在管理用户端的硬件设备！也就是说：“每部用户端主机都需要安装 X Server，而服务器端则是提供 X Client 软件， 以提供用户端绘图所需要的数据数据”。

X Server / X Client 的互动并非仅有 client --> server，两者其实有互动的！从上图 23.1.1 我们也可以发现， X Server 还有一个重要的工作，那就是将来自输入设备 （如键盘、鼠标等） 的动作告知 X Client， 你晓得， X Server 既然是管理这些周边硬件，所以，周边硬件的动作当然是由 X Server 来管理的， 但是 X Server 本身并不知道周边设备这些动作会造成什么显示上的效果， 因此 X Server 会将周边设备的这些动作行为告知 X Client ，让 X Client 去伤脑筋。

- X Client：负责 X Server 要求的“事件”之处理：

前面提到的 X Server 主要是管理显示接口与在屏幕上绘图，同时将输入设备的行为告知 X Client， 此时 X Client 就会依据这个输入设备的行为来开始处理，最后 X Client 会得到“ 嗯！这个输入设备的行为会产生某个图示”，然后将这个图示的显示数据回传给 X Server ， X server 再根据 X Client 传来的绘图数据将他描图在自己的屏幕上，来得到显示的结果。

也就是说， X Client 最重要的工作就是处理来自 X Server 的动作，将该动作处理成为绘图数据， 再将这些绘图数据传回给 X Server 啰！由于 X Client 的目的在产生绘图的数据，因此我们也称呼 X Client 为 X Application （X 应用程序）。而且，每个 X Client 并不知道其他 X Client 的存在， 意思是说，如果有两个以上的 X client 同时存在时，两者并不知道对方到底传了什么数据给 X Server ， 因此 X Client 的绘图常常会互相重叠而产生困扰喔！

举个例子来说，当我们在 X Window 的画面中，将鼠标向右移动，那他是怎么告知 X Server 与 X Client 的呢？ 首先， X server 会侦测到鼠标的移动，但是他不知道应该怎么绘图啊！此时，他将鼠标的这个动作告知 X Client， X Client 就会去运算，结果得到，嘿嘿！其实要将鼠

标指标向右移动几个像素，然后将这个结果告知 X server ， 接下来，您就会看到 X Server 将鼠标指标向右移动啰～

这样做有什么好处啊？最大的好处是， X Client 不需要知道 X Server 的硬件配备与操作系统！因为 X Client 单纯就是在处理绘图的数据而已，本身是不绘图的。所以，在用户端的 X Server 用的是什么硬件？用的是哪套操作系统？服务器端的 X Client 根本不需要知道～相当的先进与优秀～对吧！ ^_^ 整个运行流程可以参考下图：用户端用的是什么操作系统在 Linux 主机端是不在乎的！

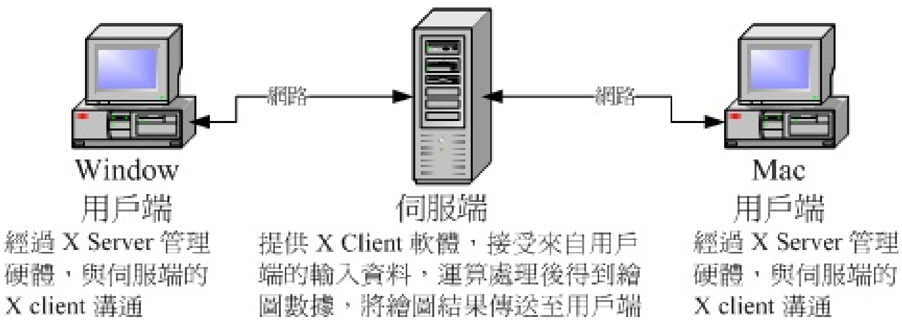


图23.1.2、X Server 用户端的操作系统与 X client 的沟通示意

- X Window Manager：特殊的 X Client ，负责管理所有的 X client 软件

刚刚前面提到，X Client 的主要工作是将来自 X Server 的数据处理成为绘图数据，再回传给 X server 而已， 所以 X client 本身是不知道他在 X Server 当中的位置、大小以及其他相关信息的。这也是上面我们谈到的， X client 彼此不知道对方在屏幕的哪个位置啊！为了克服这个问题，因此就有 Window Manager （WM, 窗口管理员） 的产生了。 窗口管理员也是 X client ，只是他主要在负责全部 X client 的控管，还包括提供某些特殊的功能，例如：

- 提供许多的控制元素，包括工作列、背景桌面的设置等等；
- 管理虚拟桌面 （virtual desktop） ；
- 提供窗口控制参数，这包括窗口的大小、窗口的重叠显示、窗口的移动、窗口的最小化等等。

我们常常听到的 KDE, GNOME, XFCE 还有阳春到爆的 twm 等等，都是一些窗口管理员的专案计划啦！ 这些专案计划中，每种窗口管理员所用以开发的显示发动机都不太相同，所著重的方向也不一样， 因此我们才会说，在 Linux 下面，每套 Window Manager 都是独特存在的，不是换了桌面与显示效果而已， 而是连显示的发动机都不会一样喔！下面是这些常见的窗口管理员全名与链接：

- GNOME （GNU Network Object Model Environment） ：http://www.gnome.org/
- KDE （K Desktop Enviroment） ：http://kde.org/
- twm （Tab Window Manager） ：http://xwinman.org/vtwm.php
- XFCE （XForms Common Environment） ：http://www.xfce.org/

由于 Linux 越来越朝向 Desktop 桌面电脑使用方向走，因此窗口管理员的角色会越来越重要！ 目前我们 CentOS 默认提供的有 GNOME 与 KDE ，这两个窗口管理员上面还有提供非常多的 X client 软件， 包括办公室生产力软件 （Open Office） 以及常用的网络功能 （firefox 浏览器、 Thunderbird 收发信件软件） 等。 现在使用者想要接触 Linux 其实真的越来越简单了，如果不要架设服务器，那么 Linux 桌面的使用与 Windows 系统可以说是一模一样的！不需要学习也能够入门哩！ ^_^

那么你知道 X Server / X client / window manager 的关系了吗？我们举 CentOS 默认的 GNOME 为例好了， 由于我们要在本机端启动 X Window system ，因此，在我们的 CentOS 主机上面必须要有 Xorg 的 X server 核心， 这样才能够提供屏幕的绘制啊～然后为了让窗口管理更方便，于是就加装了 GNOME 这个计划的 window manager ， 然后为了让自己的使用更方便，于是就在 GNOME 上面加上更多的窗口应用软件，包括输入法等等的， 最后就建构出我们的 X Window System 啰～ ^_^！所以你也会知道，X server/X client/Window Manager 是同时存在于我们一部 Linux 主机上头的啦！

- Display Manager：提供登陆需求

谈完了上述的数据后，我们得要了解一下，那么我如何取得 X Window 的控制？在本机的命令行下面你可以输入 startx 来启动 X 系统，此时由于你已经登陆系统了，因此不需要重新登陆即可取得 X 环境。但如果是 graphical.target 的环境呢？你会发现在 tty1 或其他 tty 的地方有个可以让你使用图形接口登陆 （输入帐号密码） 的咚咚，那个是啥？ 是 X Server/X client 还是什么的？其实那是个 Display Manager 啦！这个 display manager 最大的任务就是提供登陆的环境， 并且载入使用者选择的 Window Manager 与语系等数据喔！

几乎所有的大型窗口管理员专案计划都会提供 display manager 的，在 CentOS 上面我们主要利用的是 GNOME 的 GNOME Display Manager （gdm） 这支程序来提供 tty1 的图形接口登陆喔！至于登陆后取得的窗口管理员， 则可以在 gdm 上面进行选择的！我们在第四章介绍的登陆环境， 那个环境其实就是 gdm 提供的啦！再回去参考看看图示吧！ ^_^！所以说，并非 gdm 只能提供 GNOME 的登陆而已喔！

23.1.3 X Window 的启动流程

现在我们知道要启动 X Window System 时，必须要先启动管理硬件与绘图的 X Server ，然后才载入 X Client 。 基本上，目前都是使用 Window Manager 来管理窗口接口风格的。那么如何取得这样的窗口系统呢？ 你可以通过登陆本机的命令行后，输入 startx 来启动 X 窗口； 也能够通过 display manager （如果有启动 graphical.target） 提供的登陆画面，输入你的帐号密码来登陆与取得 X 窗口的！

问题是，你的 X server 配置文件为何？如何修改分辨率与显示器？你能不能自己设置默认启动的窗口管理员？ 如何设置默认的使用者环境 （与 X client 有关） 等等的，这些数据都需要通过了解 X 的启动流程才能得知！ 所以，下面我们就来谈谈如何启动 X 的流程吧！ ^_^

- 在命令行启动 X ：通过 startx 指令

我们都知道 Linux 是个多用户多任务的操作系统，所以啦，X 窗口也是可以根据不同的使用者而有不同的设置！ 这也就是说，每个用户启动 X 时， X server 的分辨率、启动 X client 的相关软件及 Window Manager 的选择可能都不一样！ 但是，如果你是首次登陆 X 呢？也就是说，你自己还没有创建自己的专属 X 画面时，系统又是从哪里给你这个 X 默认画面呢？而如果你已经设置好相关的信息，这些信息又是存放于何处呢？

事实上，当你在纯命令行且并没有启动 X 窗口的情况下来输入 startx 时，这个 startx 的作用就是在帮你设置好上头提到的这些动作啰！ startx 其实是一个 shell script ，他是一个比较友好的程序，会主动的帮忙使用者创建起他们的 X 所需要引用的配置文件而已。你可以自行研究一下 startx 这个 script 的内容，鸟哥在这里仅就 startx 的作用作个介绍。

startx 最重要的任务就是找出使用者或者是系统默认的 X server 与 X client 的配置文件，而使用者也能够使用 startx 外接参数来取代配置文件的内容。这个意思是说：startx 可以直接启动，也能够外接参数，例如下面格式的启动方式：

```
[root@study ~]# startx [X client 参数] -- [X server 参数]

# 范例：以色彩深度为 16 bit 启动 X
[root@study ~]# startx  --  -depth 16
```

startx 后面接的参数以两个减号“--”隔开，前面的是 X Client 的设置，后面的是 X Server 的设置。 上面的范例是让 X server 以色彩深度 16 bit 色 （亦即每一像素占用 16 bit ，也就是 65536 色） 显示， 因为色彩深度是与 X Server 有关的，所以参数当然是写在 -- 后面啰，于是就成了上面的模样！

你会发现，鸟哥上面谈到的 startx 都是提到如何找出 X server / X client 的设置值而已！ 没错，事实上启动 X 的是 xinit 这支程序， startx 仅是在帮忙找出设置值而已！那么 startx 找到的设置值可用顺序为何呢？基本上是这样的：

- X server 的参数方面：
    1. 使用 startx 后面接的参数；
    2. 若无参数，则找寻使用者主文件夹的文件，亦即 ~/.xserverrc
    3. 若无上述两者，则以 /etc/X11/xinit/xserverrc
    4. 若无上述三者，则单纯执行 /usr/bin/X （此即 X server 可执行文件）
- X client 的参数方面：
    1. 使用 startx 后面接的参数；
    2. 若无参数，则找寻使用者主文件夹的文件，亦即 ~/.xinitrc
    3. 若无上述两者，则以 /etc/X11/xinit/xinitrc
    4. 若无上述三者，则单纯执行 xterm （此为 X 下面的终端机软件）

根据上述的流程找到启动 X 时所需要的 X server / X client 的参数，接下来 startx 会去调用 xinit 这支程序来启动我们所需要的 X 窗口系统整体喔！接下来当然就是要谈谈 xinit 啰～

- 由 startx 调用执行的 xinit

事实上，当 startx 找到需要的设置值后，就调用 xinit 实际启动 X 的。他的语法是：

```
[root@study ~]# xinit [client option] -- [server or display option]
```

那个 client option 与 server option 如何下达呢？其实那两个咚咚就是由刚刚 startx 去找出来的啦！ 在我们通过 startx 找到适当的 xinitrc 与 xserverrc 后，就交给 xinit 来执行。 在默认的情况下 （使用者尚未有 ~/.xinitrc 等文件时），你输入 startx ， 就等于进行 xinit /etc/X11/xinit/xinitrc -- /etc/X11/xinit/xserverrc 这个指令一般！但由于 xserverrc 也不存在，参考上一小节的参数搜寻顺序， 因此实际上的指令是：xinit /etc/X11/xinit/xinitrc -- /usr/bin/X，这样瞭了吗？

那为什么不要直接执行 xinit 而是使用 startx 来调用 xinit 呢？这是因为我们必须要取得一些参数嘛！ startx 可以帮我们快速的找到这些参数而不必手动输入的。因为单纯只是执行 xinit 的时候，系统的默认 X Client 与 X Server 的内容是这样的：[3]

```
xinit xterm  -geometry  +1+1  -n  login  -display  :0 --  X  :0
```

在 X client 方面：那个 xterm 是 X 窗口下面的虚拟终端机，后面接的参数则是这个终端机的位置与登陆与否。 最后面会接一个“ -display :0 ”表示这个虚拟终端机是启动在“第 :0 号的 X 显示接口”的意思。至于 X Server 方面， 而我们启动的 X server 程序就是 X 啦！其实 X 就是 Xorg 的链接文件，亦即是 X Server 的主程序啰！ 所以我们启动 X 还挺简单的～直接执行 X 而已，同时还指定 X 启动在第 :0 个 X 显示接口。 如果单纯以上面的内容来启动你的 X 系统时，你就会发现 tty2 以后的终端机有画面了！只是.....很丑～因为我们还没有启动 window manager 啊！

从上面的说明我们可以知道， xinit 主要在启动 X server 与载入 X client ，但这个 xinit 所需要的参数则是由 startx 去帮忙找寻的。因此，最重要的当然就是 startx 找到的那些参数啦！ 所以呢，重点当然就是 /etc/X11/xinit/ 目录下的 xinitrc 与 xserverrc 这两个文件的内容是啥啰～ 虽然 xserverrc 默认是不存在的。下面我们就分别来谈一谈这两个文件的主要内容与启动的方式～

- 启动 X server 的文件： xserverrc

X 窗口最先需要启动的就是 X server 啊，那 X server 启动的脚本与参数是通过 /etc/X11/xinit/ 里面的 xserverrc 。不过我们的 CentOS 7.x 根本就没有 xserverrc 这个文件啊！ 那使用者主文件夹目前也没有 ~/.xserverrc ，这个时候系统会怎么做呢？其实就是执行 /usr/bin/X 这个指令啊！ 这个指令也是系统最原始的 X server 可执行文件啰。

在启动 X Server 时，Xorg 会去读取 /etc/X11/xorg.conf 这个配置文件。针对这个配置文件的内容， 我们会在下个小节介绍。如果一切顺利，那么 X 就会顺利的在 tty2 以后终端环境中启动了 X 。 单纯的 X 启动时，你只会看到画面一片漆黑，然后中心有个鼠标的光标而已～

由前一小节的说明中，你可以发现到其实 X 启动的时候还可以指定启动的接口喔！那就是 :0 这个参数，这是啥？ 事实上我们的 Linux 可以“同时启动多个 X”喔！第一个 X 的画面会在 :0 亦即是 tty2 ，第二个 X 则是 :1 亦即是 tty3 。 后续还可以有其他的 X 存在的。因此，上一小节我们也有发现， xterm 在载入时，也必须要使用 -display 来说明， 这个 X 应用程序是需要在哪个 X 载入的才行呢！其中比较有趣的是， X server 未注明载入的接口时，默认是使用 :0 ～ 但是 X client 未注明时，则无法执行喔！



Tips CentOS 7 的 tty 非常有趣！如果你在分析 systemd 的章节中有仔细看的话， 会发现到其实 tty 是有用到才会启动的，这与之前 CentOS 6 以前的版本默认启用 6 个 tty 给你是不同的。因此，如果你只有用到 tty1 的话， 那么启动 X 就会默认丢到 tty2 ，而 X :1 就会丢到 tty3 这样～以此类推喔～

启动了 X server 后，接下来就是载入 X client 到这个 X server 上面啦！

- 启动 X Client 的文件： xinitrc

假设你的主文件夹并没有 ~/.xinitrc ，则此时 X Client 会以 /etc/X11/xinit/xinitrc 来作为启动 X Client 的默认脚本。xinitrc 这个文件会将很多其他的文件参数引进来， 包括 /etc/X11/xinit/xinitrc-common 与 /etc/X11/xinit/Xclients 还有 /etc/sysconfig/desktop 。你可以参考 xinitrc 后去搜寻各个文件来了解彼此的关系。

不过分析到最后，其实最终就是载入 KDE 或者是 GNOME 而已。你也可以发现最终在 XClient 文件当中会有两个指令的搜寻， 包括 startkde 与 gnome-session 这两个，这也是 CentOS 默认会提供的两个主要的 Window Manager 啰。 而你也可以通过修改 /etc/sysconfig/desktop 内的 DESKTOP=GNOME 或 DESKTOP=KDE 来决定默认使用哪个窗口管理员的。 如果你并没有安装这两个大家伙，那么 X 就会去使用阳春的 twm 这个窗口管理员来管理你的环境啰。



Tips 不论怎么说，鸟哥还是希望大家可以通过解析 startx 这个 script 的内容去找到每个文件， 再根据分析每个文件来找到您 distributions 上面的 X 相关文件～ 毕竟每个版本的 Linux 还是有所差异的～

另外，如果有特殊需求，你当然可以自订 X client 的参数！这就得要修改你主文件夹下的 ~/.xinitrc 这个文件啰。 不过要注意的是，如果你的 .xinitrc 配置文件里面有启动的 x client 很多的时候，千万注意将除了最后一个 window manager 或 X Client 之外，都放到背景里面去执行啊！举例来说，像下面这样：

```
xclock -geometry 100x100-5+5 &
      xterm -geometry 80x50-50+150 &
      exec /usr/bin/twm
```

意思就是说，我启动了 X ，并且同时启动 xclock / xterm / twm 这三个 X clients 喔！ 如此一来，你的 X 就有这三个咚咚可以使用了！如果忘记加上 & 的符号，那就..... 会让系统等待啊，而无法一次就登陆 X 呢！

- X 启动的端口

好了，根据上面的说明，我们知道要在命令行下面启动 X 时，直接使用 startx 来找到 X server 与 X client 的参数或配置文件， 然后再调用 xinit 来启动 X 窗口系统。xinit 先载入 X server 到默认的 :0 这个显示接口，然后再载入 X client 到这个 X 显示接口上。而 X client 通常就是 GNOME 或 KDE ，这两个设置也能够在 /etc/sysconfig/desktop 里面作好设置。最后我们想要了解的是，既然 X 是可以跨网络的，那 X 启动的端口是几号？

其实，CentOS 由于考虑 X 窗口是在本机上面运行，因此将端口改为插槽档 （socket） 了，因此你无法观察到 X 启动的端口的。事实上， X server 应该是要启动一个 port 6000 来与 X client 进行沟通的！ 由于系统上面也可能有多个 X 存在，因此我们就会有 port 6001, port 6002... 等等。这也就是说：（假设为 multi-user.target 模式， 且用户仅曾经切换到 tty1 而已）

X 窗口系统	显示接口号码	默认终端机	网络监听端口
第一个 X	hostname:0	tty2	port 6000
第二个 X	hostname:1	tty3	port 6001

在 X Window System 的环境下，我们称 port 6000 为第 0 个显示接口，亦即为 hostname:0 ， 那个主机名称通常可以不写，所以就成了 :0 即可。在默认的情况下，第一个启动的 X （不论是启动在第几个 port number） 是在 tty2 ，亦即按下 [ctrl]+[Alt]+[F2] 那个画面。 而起动的第二个 X （注意到了吧！可以有多个 X 同时启动在您的系统上呢） 则默认在 tty3 亦即 [ctrl]+[Alt]+[F3] 那个画面呢！很神奇吧！ ^_^

如前所述，因为主机上的 X 可能有多个同时存在，因此，当我们在启动 X Server / Client 时， 应该都要注明该 X Server / Client 主要是提供或接受来自哪个 display 的 port number 才行。

23.1.4 X 启动流程测试

好了，我们可以针对 X Server 与 X client 的架构来做个简单的测试喔！ 这里鸟哥假设你的 tty1 是 multi-user.target 的，而且你也曾经在 tty2 测试过相关的指令，所以你的 X :1 将会启用在 tty3 喔！ 而且，下面的指令都是在 tty1 的地方执行的，至于下面的画面则是在 tty3 的地方展现。 因此，请自行切换 tty1 下达指令与 tty3 查阅结果啰！

```
1\. 先来启动第一个 X 在 :1 画面中：
[dmtsai@study ~]$ X :1 &
```


图23.1.3、单纯启动 X server的情况

上述的 X 是大写，那个 :1 是写在一起的，至于 & 则是放到背景去执行。此时系统会主动的跳到第二个图形接口终端机，亦即 tty8 上喔！所以如果一切顺利的话，你应该可以看到一个 X 的鼠标光标可以让你移动了（如上图所示）。 该画面就是 X Server 启动的画面啰！丑丑的，而且没有什么 client 可以用啊！ 接下来，请按下 [ctrl]+[alt]+[F1] 回到刚刚下达指令的终端机：（若没有 xterm 请自行 yum 安装它！）

```
2\. 输入数个可以在 X 当中执行的虚拟终端机
[dmtsai@study ~]$ xterm -display :1  &
[dmtsai@study ~]$ xterm -display :1  &
```



图23.1.4、在 X 上面启动 xterm 终端机显示的结果

那个 xterm 是必须要在 X 下面才能够执行的终端机接口。加入的参数 -display 则是指出这个 xterm 要在那个 display 使用的。这两个指令请不要一次下完！先执行一次，然后按下 [ctrl]+[alt]+[F3] 去到 X 画面中，你会发现多了一个终端机啰～ 不过，可惜的是，你无法看到终端机的标题、也无法移动终端机，当然也无法调整终端机的大小啊！我们回到刚刚的 tty1 然后再次下达 xterm 指令，理论上应该多一个终端机，去到 tty3 查阅一下。唉～没有多出一个终端机啊？ 这是因为两个终端机重叠了～我们又无法移动终端机，所以只看到一个。 接下来，请再次回到 tty1 去下达指令吧！（可能需要 yum install xorg-x11-apps 喔！）

```
3\. 在输入不同的 X client 观察观察，分别去到 tty3 观察喔！
[dmtsai@study ~]$ xclock -display :1  &
[dmtsai@study ~]$ xeyes -display :1  &
```

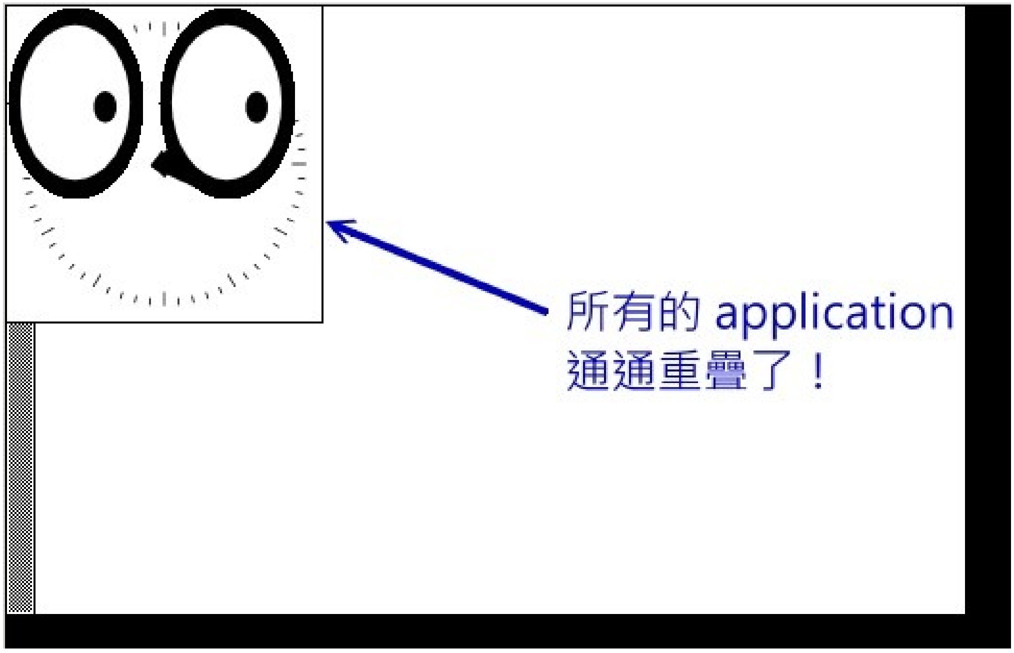


图23.1.5、分别启动 xclock 时钟与 xeyes 眼睛的结果

跟前面一样的，我们又多执行了两个 X client ，其中 xclock 会显示时钟，而 xeyes 则是会出现一双大眼睛来盯着光标！ 你可以移动一下光标就可以发现眼睛的焦聚会跑啊 ^_^！不过，目前的四个 X client 通通不能够移动与放大缩小！ 如此一来，你怎么在 xterm 下面下达指令啊？当然就很困扰～所以让我们来载入最阳春的窗口管理员吧！

```
4\. 输入可以管理的 window manager，我们这边先以 root 来安装 twm 喔！
[root@study ~]# yum install http://ftp.ksu.edu.tw/FTP/CentOS/6/os/x86_64/\
&gt; Packages/xorg-x11-twm-1.0.3-5.1.el6.x86_64.rpm
# 真要命！CentOS 7 说 twm 已经没有在维护，所以没有提供这玩意儿了！鸟哥只好拿旧版的 twm 来安装！
# 请您自行到相关的网站上找寻这个 twm 啰！因为版本可能会不一样！
[root@study ~]# yum install xorg-x11-fonts-{100dpi,75dpi,Type1}

5\. 接下来就可以开始用 dmtsai 的身份来玩一下这玩意儿了！
[dmtsai@study ~]$ twm -display :1  &
```

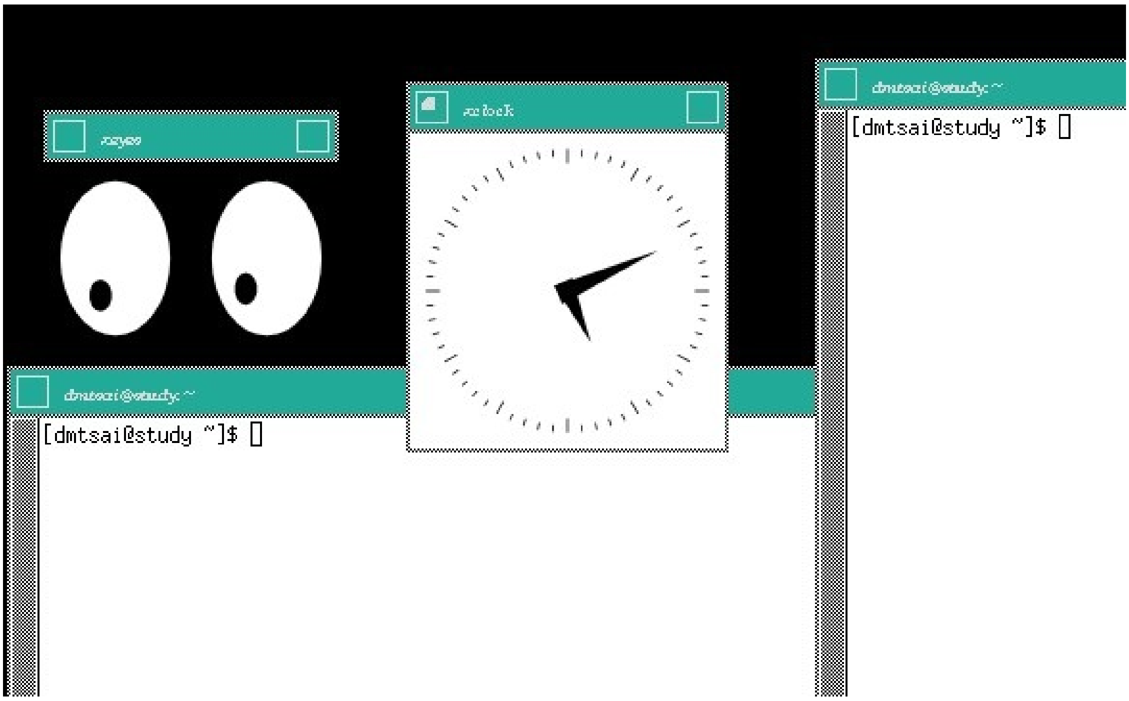


图 23.1.6、窗口管理员 twm 的功能显示

回到 tty1 后，用最简单的 twm 这个窗口管理员来管理我们的 X 吧！输入之后，去到 tty3 看看，用鼠标移动一下终端机看看？可以移动了吧？也可以缩小放大窗口啰～同时也出现了标题提示啰～也看到两个终端机啦！ 现在终于知道窗口管理员的重要性了吧？^_^！在黑屏幕地方按下鼠标右键，就会出现类似上面画面最右边的菜单， 你就可以进行额外的管理啰～玩玩看先！

```
6\. 将所有刚刚创建的 X 相关工作全部杀掉！
[dmtsai@study ~]# kill %6 %5 %4 %3 %2 %1
```

很有趣的一个小实验吧～通过这个实验，你应该会对 X server 与 Window manager 及 tty3 以后的终端接口使用方式有比较清楚的了解～加油！

23.1.5 我是否需要启用 X Window System

谈了这么多 X 窗口系统方面的信息后，再来聊聊，那么你的 Linux 主机是否需要默认就启动 X 窗口呢？ 一般来说，如果你的 Linux 主机定位为网络服务器的话，那么由于 Linux 里面的主要服务的配置文件都是纯文本的格式文件， 相当的容易设置的，所以啊，根本就是不需要 X Window 存在呢！因为 X Window 仅是 Linux 系统内的一个软件而已啊！

但是万一你的 Linux 主机是用来作为你的桌上计算机用的，那么 X Window 对你而言，就是相当重要的一个咚咚了！因为我们日常使用的办公室软件，都需要使用到 X Window 图形的功能呢！此外，以鸟哥的例子来说，俺之前接触到的数值分析模式，需要利用图形处理软件来将数据读取出来， 所以在那部 Linux 主机上面，我一定需要 X Window 的。

由于目前的主机系统配备已经很不错，除非你使用的是单版计算机，否则桌面电脑、笔记本电脑的系统配备要拿来跑 X window 大概都不是问题！ 所以，是否默认要启用你的 X window 系统，完全掌握在你的服务器用途考虑上啰！！

23.2 X Server 配置文件解析与设置

从前面的说明来看，我们知道一个 X 窗口系统能不能成功启动，其实与 X Server 有很大的关系的。因为 X Server 负责的是整个画面的描绘，所以没有成功启动 X Server 的话，即使有启动 X Client 也无法将图样显示出来啊。所以，下面我们就针对 X Server 的配置文件来做个简单的说明，好让大家可以成功的启动 X Window System 啊。

基本上， X Server 管理的是显卡、屏幕分辨率、鼠标按键对应等等，尤其是显卡芯片的认识，真是重要啊。 此外，还有显示的字体也是 X Server 管理的一环。基本上，X server 的配置文件都是默认放置在 /etc/X11 目录下，而相关的显示模块或上面提到的总总模块，则主要放置在 /usr/lib64/xorg/modules 下面。比较重要的是字体文件与芯片组，她们主要放置在:

- 提供的屏幕字体: /usr/share/X11/fonts/
- 显卡的芯片组: /usr/lib64/xorg/modules/drivers/

在 CentOS 下面，这些都要通过一个统一的配置文件来规范，那就是 X server 的配置文件啦。这个配置文件的文件名就是 /etc/X11/xorg.conf 喔！

23.2.1 解析 xorg.conf 设置

如同前几个小节谈到的，在 Xorg 基金会里面的 X11 版本为 X11R7.N ，那如果你想要知道到底你用的 X Server 版本是第几版，可以使用 X 指令来检查喔！（你必须以 root 的身分执行下列指令）

```
[root@study ~]# X -version
X.Org X Server 1.15.0
Release Date: 2013-12-27
X Protocol Version 11, Revision 0
Build Operating System:  2.6.32-220.17.1.el6.x86_64
Current Operating System: Linux study.centos.vbird 3.10.0-229.el7.x86_64 #1 SMP Fri Mar
  6 11:36:42 UTC 2015 x86_64
Kernel command line: BOOT_IMAGE=/vmlinuz-3.10.0-229.el7.x86_64 root=/dev/mapper/centos-
  root ro rd.lvm.lv=centos/root rd.lvm.lv=centos/swap crashkernel=auto rhgb quiet
Build Date: 10 April 2015  11:44:42AM
Build ID: xorg-x11-server 1.15.0-33.el7_1
Current version of pixman: 0.32.4
        Before reporting problems, check http://wiki.x.org
        to make sure that you have the latest version.
```

由上面的几个关键字我们可以知道，目前鸟哥的这部测试机使用的 X server 是 Xorg 计划所提供的 X11 版， 不过看起来 Xorg 已经将所谓的 X11R7 那个 R7 的版次移除，使用的是 Xorg 自己的版次了！所以是 Xorg 1.15.0 版本！ 此外，若有问题则可以到 http://wiki.x.org 去查询～因为是 Xorg 这个 X server ，因此我们的配置文件文件名为 /etc/X11/xorg.conf 这一个哩。所以，理解这个文件的内容对于 X server 的功能来说，是很重要的。

比较需要留意的是，从 CentOS 6 以后 （当然包含 CentOS 7），X server 在每次启动的时候都会自行侦测系统上面的显示芯片、屏幕类型等等， 然后自行搭配最优化的驱动程序载入。因此，这个 /etc/X11/xorg.conf 已经不再被需要了。不过，如果你不喜欢 X 系统自行侦测的设置值， 那也可以自行创建 xorg.conf 就是了。

此外，如果你只想要加入或者是修改部份的设置，并不是每个元件都要自行设置的话，那么可以在 /etc/X11/xorg.conf.d/ 这个目录下创建文件名为 .conf 的文件， 将你需要的额外项目加进去即可喔！那就不会每个设置都以你的 xorg.conf 为主了！了解乎？



Tips 那我怎么知道系统用的是哪一个设置呢？可以参考 /var/log/Xorg.0.log 的内容，该文件前几行会告诉你使用的设置文件是来自于哪里的喔！

注意一下，在修改这个文件之前，务必将这个文件给它备份下来，免的改错了什么东西导致连 X server 都无法启动的问题啊。这个文件的内容是分成数个段落的，每个段落以 Section 开始，以 EndSection 结束， 里面含有该 Section （段落） 的相关设置值，例如:

```
Section  "section name"
…… &lt;== 与这个 section name 有关的设置项目
……
EndSection
```

至于常见的 section name 主要有:

1. Module: 被载入到 X Server 当中的模块 （某些功能的驱动程序）；
2. InputDevice: 包括输入的 1. 键盘的格式 2. 鼠标的格式，以及其他相关输入设备；
3. Files: 设置字体所在的目录位置等；
4. Monitor: 监视器的格式， 主要是设置水平、垂直的更新频率，与硬件有关；
5. Device: 这个重要，就是显卡芯片组的相关设置了；
6. Screen: 这个是在屏幕上显示的相关分辨率与色彩深度的设置项目，与显示的行为有关；
7. ServerLayout: 上述的每个项目都可以重复设置，这里则是此一 X server 要取用的哪个项目值的设置啰。

前面说了，xorg.conf 这个文件已经不存在，那我们怎么学习呢？没关系，Xorg 有提供一个简单的方式可以让我们来重建这个 xorg.conf 文件！ 同时，这可能也是 X 自行侦测 GPU 所产生的最优化设置喔！怎么处理呢？假设你是在 multi-user.target 的环境下，那就可以这样作来产生 xorg.conf 喔！

```
[root@study ~]# Xorg -configure
.....（前面省略）.....
Markers: （--） probed, （**） from config file, （==） default setting,
        （++） from command line, （!!） notice, （II） informational,
        （WW） warning, （EE） error, （NI） not implemented, （??） unknown.
（==） Log file: "/var/log/Xorg.0.log", Time: Wed Sep 16 10:13:57 2015
List of video drivers:   # 这里在说明目前这个系统上面有的显卡芯片组的驱动程序有哪些的意思
        qxl
        vmware
        v4l
        ati
        radeon
        intel
        nouveau
        dummy
        modesetting
        fbdev
        vesa
（++） Using config file: "/root/xorg.conf.new"        # 使用的配置文件
（==） Using config directory: "/etc/X11/xorg.conf.d"  # 额外设置项目的位置
（==） Using system config directory "/usr/share/X11/xorg.conf.d"
（II） [KMS] Kernel modesetting enabled.

.....（中间省略）.....

Your xorg.conf file is /root/xorg.conf.new           # 最终新的文件出现了！

To test the server, run 'X -config /root/xorg.conf.new' # 测试手段！
```

这样就在你的 root 主文件夹产生一个新的 xorg.conf.new 啰！好了，直接来看看这个文件的内容吧！这个文件默认的情况是取消很多设置值的， 所以你的配置文件可能不会看到这么多的设置项目。不要紧的，后续的章节会交代如何设置这些项目的喔！

```
[root@study ~]# vim xorg.conf.new
Section "ServerLayout"                                # 目前 X 决定使用的设置项目
        Identifier     "X.org Configured"
        Screen      0  "Screen0" 0 0                  # 使用的屏幕为 Screen0 这一个 （后面会解释）
        InputDevice    "Mouse0" "CorePointer"         # 使用的鼠标设置为 Mouse0
        InputDevice    "Keyboard0" "CoreKeyboard"     # 使用的键盘设置为 Keyboard0
EndSection
# 系统可能有多组的设置值，包括多种不同的键盘、鼠标、显示芯片等等，而最终 X 使用的设置，
# 就是在这个 ServerLayout 项目中来处理的！因此，你还得要去下面找出 Screen0 是啥

Section "Files"
        ModulePath   "/usr/lib64/xorg/modules"
        FontPath     "catalogue:/etc/X11/fontpath.d"
        FontPath     "built-ins"
EndSection
# 我们的 X Server 很重要的一点就是必须要提供字体，这个 Files 的项目就是在设置字体，
# 当然啦，你的主机必须要有字体文件才行。一般字体文件在：/usr/share/X11/fonts/ 目录中。
# 但是 Xorg 会去读取的则是在 /etc/X11/fontpath.d 目录下的设置喔！

Section "Module"
        Load  "glx"
EndSection
# 上面这些模块是 X Server 启动时，希望能够额外获得的相关支持的模块。
# 关于更多模块可以搜寻一下 /usr/lib64/xorg/modules/extensions/ 这个目录

Section "InputDevice"
        Identifier  "Keyboard0"
        Driver      "kbd"
EndSection
# 就是键盘，在 ServerLayout 项目中有出现这个 Keyboard0 吧！主要是设置驱动程序！

Section "InputDevice"
        Identifier  "Mouse0"
```

```
        Driver      "mouse"
        Option      "Protocol" "auto"
        Option      "Device" "/dev/input/mice"
        Option      "ZAxisMapping" "4 5 6 7"   # 支持滚轮功能！
EndSection
# 这个则主要在设置鼠标功能，重点在那个 Protocol 项目，
# 那个是可以指定鼠标接口的设置值，我这里使用的是自动侦测！不论是 USB/PS2。

Section "Monitor"
        Identifier   "Monitor0"
        VendorName   "Monitor Vendor"
        ModelName    "Monitor Model"
EndSection
# 屏幕监视器的设置仅有一个地方要注意，那就是垂直与水平的更新频率，常见设置如下：
#       HorizSync    30.0 - 80.0
#       VertRefresh  50.0 - 100.0
# 在上面的 HorizSync 与 VerRefresh 的设置上，要注意，不要设置太高，
# 这个玩意儿与实际的监视器功能有关，请查询你的监视器手册说明来设置吧！
# 传统 CRT 屏幕设置太高的话，据说会让 monitor 烧毁呢，要很注意啊。

Section "Device"     # 显卡芯片 （GPU） 的驱动程序！很重要的设置！
        Identifier  "Card0"
        Driver      "qxl"        # 实际使用的显卡驱动程序！
        BusID       "PCI:0:2:0"
EndSection
# 这地方重要了，这就是显卡的芯片模块载入的设置区域。由于鸟哥使用 Linux KVM
# 仿真器仿真这个测试机，因此这个地方显示的驱动程序为 qxl 模块。
# 更多的显示芯片模块可以参考 /usr/lib64/xorg/modules/drivers/

Section "Screen"                # 与显示的画面有关，分辨率与色彩深度
        Identifier "Screen0"    # 就是 ServerLayout 里面用到的那个屏幕设置
        Device     "Card0"      # 使用哪一个显卡的意思！
        Monitor    "Monitor0"   # 使用哪一个屏幕的意思！
        SubSection "Display"    # 此阶段的附属设置项目
                Viewport   0 0
                Depth     1     # 就是色彩深度的意思！
        EndSubSection
        SubSection "Display"
                Viewport   0 0
                Depth     16
        EndSubSection
        SubSection "Display"
                Viewport   0 0
                Depth     24
        EndSubSection
EndSection
# Monitor 与实际的显示器有关，而 Screen 则是与显示的画面分辨率、色彩深度有关。
# 我们可以设置多个分辨率，实际应用时可以让使用者自行选择想要的分辨率来呈现，设置如下：
#         Modes    "1024x768" "800x600" "640x480" &lt;==分辨率
# 上述的 Modes 是在 "Display" 下面的子设置。
# 不过，为了避免困扰，鸟哥通常只指定一到两个分辨率而已。
```

上面设置完毕之后，就等于将整个 X Server 设置妥当了，很简单吧。如果你想要更新其他的例如显示芯片的模块的话，就得要去硬件开发商的网站下载原始文件来编译才行。 设置完毕之后，你就可以启动 X Server 试看看啰。然后，请将 xorg.conf.new 更名成类似 00-vbird.conf 之类的文件名， 再将该文件移动到 /etc/X11/xorg.conf.d/ 里面去，这样就 OK 了！

```
# 测试 X server 的配置文件是否正常：
[root@study ~]# startx    &lt;==直接在 multi-user.target 启动 X 看看
[root@study ~]# Xorg :1   &lt;==在 tty3 单独启动 X server 看看
```

当然，你也可以利用 systemctl isolate graphical.target 这个指令直接切换到图形接口的登陆来试看看啰。



Tips 经由讨论区网友的说明，如果你发现明明有捉到显卡驱动程序却老是无法顺利启动 X 的话，可以尝试去官网取得驱动程序来安装， 也能够将“Device”阶段的“Driver”修改成默认的“Driver "vesa"”，使用该驱动程序来暂时启动 X 内的显卡喔！

23.2.2 字体管理

我们 Xorg 所使用的字体大部分都是放置于下面的目录中：

- /usr/share/X11/fonts/
- /usr/share/fonts/

不过 Xorg 默认会载入的字体则是记录于 /etc/X11/fontpath.d/ 目录中，使用链接文件的模式来进行链接的动作而已。 你应该还记得 xorg.conf 里面有个“ Flies ”的设置项目吧？该项目里面就有指定到“ FontPath "catalogue:/etc/X11/fontpath.d" ” 对吧！也就是说，我们默认的 Xorg 使用的字体就是取自于 /etc/X11/fontpath.d 啰！

鸟哥查了一下 CentOS 7 针对中文字体 （chinese） 来说，有楷书与明体，明体默认安装了，不过楷书却没有安装耶～ 那我们能不能安装了楷书之后，将楷书也列为默认的字体之一呢？来瞧一瞧我们怎么作的好了：

```
# 1\. 检查中文字体，并且安装中文字体与检验有没有放置到 fontpath.d 目录中！
[root@study ~]# ll -d /usr/share/fonts/cjk*
drwxr-xr-x. 2 root root 22 May  4 17:54 /usr/share/fonts/cjkuni-uming

[root@study ~]# yum install cjkuni-ukai-fonts
[root@study ~]# ll -d /usr/share/fonts/cjk*
drwxr-xr-x. 2 root root 21 Sep 16 11:48 /usr/share/fonts/cjkuni-ukai  # 这就是楷书！
drwxr-xr-x. 2 root root 22 May  4 17:54 /usr/share/fonts/cjkuni-uming

[root@study ~]# ll /etc/X11/fontpath.d/ lrwxrwxrwx. 1 root root 29 Sep 16 11:48 cjkuni-uk
lrwxrwxrwx. 1 root root 30 May  4 17:54 cjkuni-uming-fonts -&gt; /usr/share/fonts/cjkuni-
lrwxrwxrwx. 1 root root 36 May  4 17:52 default-ghostscript -&gt; /usr/share/fonts/defaul
lrwxrwxrwx. 1 root root 30 May  4 17:52 fonts-default -&gt; /usr/share/fonts/default/Type
lrwxrwxrwx. 1 root root 27 May  4 17:51 liberation-fonts -&gt; /usr/share/fonts/liberatio
lrwxrwxrwx. 1 root root 27 Sep 15 17:10 xorg-x11-fonts-100dpi:unscaled:pri=30 -&gt; /usr/
lrwxrwxrwx. 1 root root 26 Sep 15 17:10 xorg-x11-fonts-75dpi:unscaled:pri=20 -&gt; /usr/s
lrwxrwxrwx. 1 root root 26 May  4 17:52 xorg-x11-fonts-Type1 -&gt; /usr/share/X11/fonts/T
# 竟然会自动的将该字体加入到 fontpath.d 当中！太好了！ ^_^

# 2\. 创建该字体的字体高速缓存数据，并检查是否真的取用了？
[root@study ~]# fc-cache -v &#124; grep ukai
/usr/share/fonts/cjkuni-ukai: skipping, existing cache is valid: 4 fonts, 0 dirs

[root@study ~]# fc-list &#124; grep ukai
/usr/share/fonts/cjkuni-ukai/ukai.ttc: AR PL UKai TW:style=Book
/usr/share/fonts/cjkuni-ukai/ukai.ttc: AR PL UKai HK:style=Book
/usr/share/fonts/cjkuni-ukai/ukai.ttc: AR PL UKai CN:style=Book
/usr/share/fonts/cjkuni-ukai/ukai.ttc: AR PL UKai TW MBE:style=Book

# 3\. 重新启动 Xorg，或者是强制重新进入 graphical.target
[root@study ~]# systemctl isolate multi-user.target; systemctl isolate graphical.target
```

如果上述的动作没有问题的话，现在你可以在图形界面下面，通过“应用程序” --> “公用程序” --> “字体检视程序”当中找到一个名为 “AR PL UKai CN, Book”字样的字体，点下去就会看到如下的图示，那就代表该字体已经可以被使用了。不过某些程序可能还得要额外的加工就是了～[4]

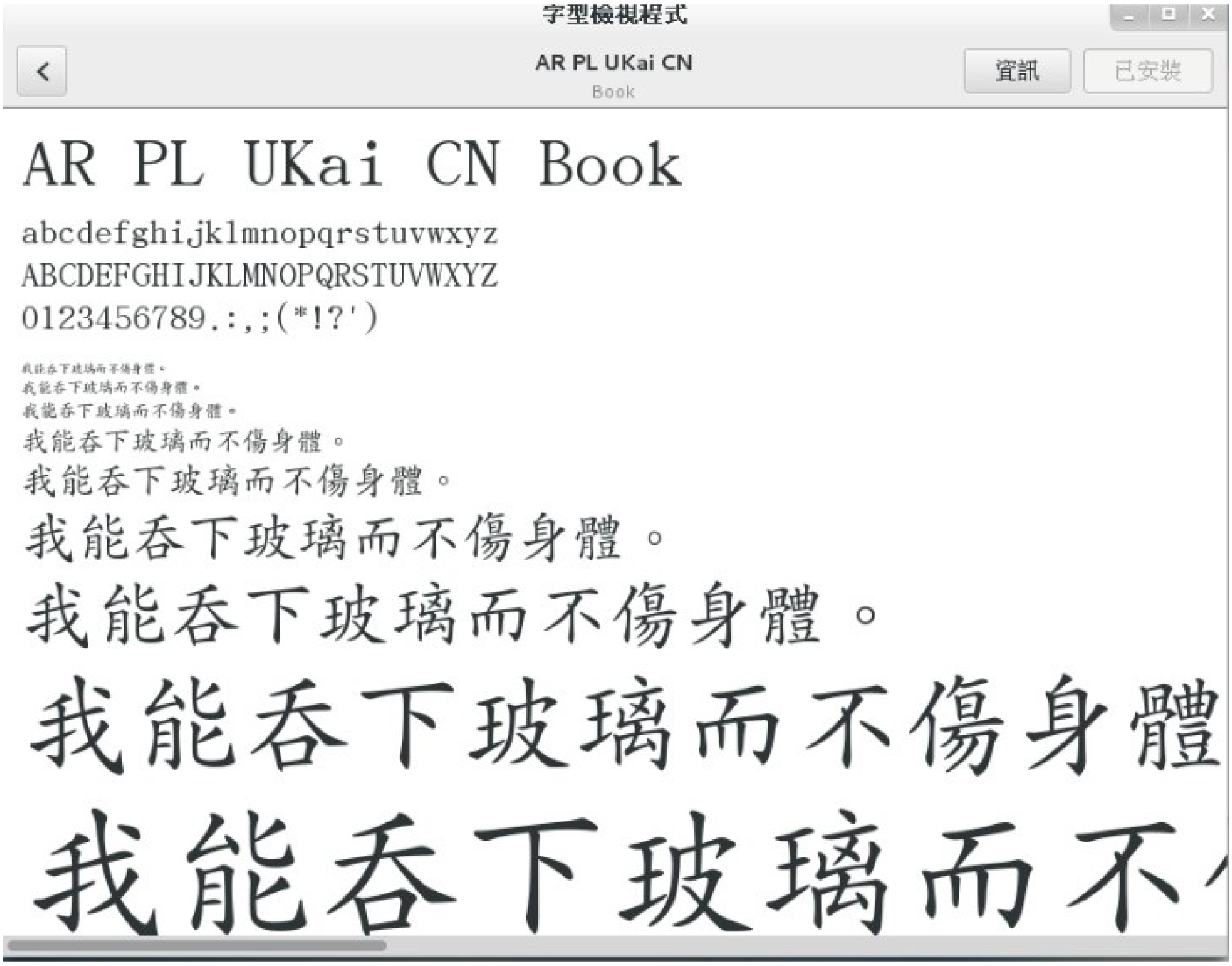


图23.2.1、安装楷书字体的结果

鸟哥比较好奇的是，这个字体的开发者怎么这么有趣！列出来的示意字体竟然是吃了玻璃会身体头好壮壮～这...会不会教坏小孩啊？呵呵呵呵～

- 让窗口管理员可以使用额外的字体

如果想要使用额外的字体的话，你可以自行取得某些字体来处理的。鸟哥这边从 Windows 微软正黑体、Times new Romans 两种字体加上粗、斜体等共六个文件来处理字体的安装～ 这边得注明一下是纯粹的测试，测试完毕后文件就给它拿掉了，并没有持续使用喔！并没有想要违法的意思啦～大家参考看看就好了。 那就来看看如何增加字体吧！（假设上述的字体文件是放置在 /root/font 中）

```
# 1\. 将字体文件放置到系统设置目录，亦即下面的目录中：
[root@study ~]# cd /usr/share/fonts/
[root@study ~]# mkdir windows
[root@study ~]# cp /root/font/*.ttf /usr/share/fonts/windows/

# 2\. 使用 fc-cache 将上述的文件加入字体的支持中：
[root@study ~]# fc-cache -f -v
....（前面省略）....
/usr/share/fonts/windows: caching, new cache contents: 6 fonts, 0 dirs
....（后面省略）....
# -v 仅是列出目前的字体数据， -f 则是强制重新创建字体高速缓存！

# 3\. 通过 fc-list 列出已经被使用的文件看看：
[root@study ~]# fc-list : file &#124; grep window  &lt;==找出被高速缓存住的文件名
/usr/share/fonts/windows/timesbi.ttf:
/usr/share/fonts/windows/timesi.ttf:
/usr/share/fonts/windows/msjh.ttf:
/usr/share/fonts/windows/times.ttf:
/usr/share/fonts/windows/msjhbd.ttf:
/usr/share/fonts/windows/timesbd.ttf:
```

之后在字体检视器里面就会发现有多了“Microsoft JhengHei, Times New Roman”等等的字体可以用啰！

23.2.3 显示器参数微调

有些朋友偶而会这样问：“我的显示器明明还不错，但是屏幕分辨率却永远只能达到 800x600 而已， 这该如何处理？”，屏幕的分辨率应该与显卡相关性不高，而是与显示器的更新频率有关！

所谓的更新频率，指的是在一段时间内屏幕重新绘制画面的速度。举例来说， 60Hz 的更新频率， 指的是每秒钟画面更新 60 次的意思。那么关于显示器的更新频率该如何调整呢？ 你得先去找到你的显示器的使用说明书 （或者是网站会有规格介绍），取得最高的更新率后，接下来选择你想要的分辨率， 然后通过这个 gtf 的指令功能来调整：



Tips 基本上，现在新的 Linux distribution 的 X server 大多使用自行侦测方式来处理所有的设置了，因此， 除非你的屏幕特别新或者是特别怪，否则应该不太需要使用到 gtf 的功能啰！

```
# 1\. 先来测试一下你目前的屏幕搭配显卡所能够处理的分辨率与更新频率 （须在 X 环境下）
[root@study ~]# xrandr
Screen 0: minimum 320 x 200, current 1440 x 900, maximum 8192 x 8192
Virtual-0 connected primary 1440x900+0+0 0mm x 0mm
   1024x768       59.9 +
   1920x1200      59.9
   1920x1080      60.0
   1600x1200      59.9
   1680x1050      60.0
   1400x1050      60.0
   1280x1024      59.9
   1440x900       59.9*
   1280x960       59.9
   1280x854       59.9
   1280x800       59.8
   1280x720       59.9
   1152x768       59.8
   800x600        59.9
   848x480        59.7
   720x480        59.7
   640x480        59.4
# 上面显示现在的环境中，测试过最高分辨率大概是 1920x1200 ，但目前是 1440x900 （*）
# 若需要调整成 1280*800 的话，可以使用下面的方式来调整喔！

[root@study ~]# xrandr -s 1280x800

# 2\. 若想强迫 X server 更改屏幕的分辨率与更新频率，则需要修订 xorg.conf 的设置。先来侦测：
[root@study ~]# gtf 水平像素 垂直像素 更新频率 [-xv]
选项与参数：
水平像素：就是分辨率的 X 轴
垂直像素：就是分辨率的 Y 轴
更新频率：与显示器有关，一般可以选择 60, 75, 80, 85 等频率
-x       ：使用 Xorg 配置文件的模式输出，这是默认值
-v       ：显示侦测的过程

# 1\. 使用 1024x768 的分辨率，75 Hz 的更新频率来取得显示器内容
[root@study ~]# gtf 1024 768 75 -x
# 1024x768 @ 75.00 Hz （GTF） hsync: 60.15 kHz; pclk: 81.80 MHz
Modeline "1024x768_75.00"  81.80  1024 1080 1192 1360  768 769 772 802  -HSync +Vsync
# 重点是 Modeline 那一行！那行给他抄下来

# 2\. 将上述的数据输入 xorg.conf.d/*.conf 内的 Monitor 项目中：
[root@study ~]# vim /etc/X11/xorg.conf.d/00-vbird.conf
Section "Monitor"
    Identifier   "Monitor0"
    VendorName   "Monitor Vendor"
    ModelName    "Monitor Model"
    Modeline "1024x768_75.00"  81.80  1024 1080 1192 1360  768 769 772 802  -HSync +Vsync
EndSection
# 就是新增上述的那行特殊字体部分到 Monitor 的项目中即可。
```

然后重新启动你的 X ，这样就能够选择新的分辨率啰！那如何重新启动 X 呢？两个方法， 一个是“ systemctl isolate multi-user.target; systemctl isolate graphical.target ”从文字模式与图形模式的执行等级去切换，另一个比较简单， 如果原本就是 graphical.target 的话，那么在 X 的画面中按下“ [alt] + [crtl] + [backspace] ”三个组合按键， 就能够重新启动 X 窗口啰！

23.3 显卡驱动程序安装范例

虽然你的 X 窗口系统已经顺利的启动了，也调整到你想要的分辨率了，不过在某些场合下面，你想要使用显卡提供的 3D 加速功能时，却发现 X 提供的默认的驱动程序并不支持！此时真是欲哭无泪啊～那该如何是好？没关系，安装官方网站提供的驱动程序即可！ 目前（2015） 世界上针对 x86 提供显卡的厂商最大的应该是 Nvidia / AMD （ATI） / Intel 这三家（没有照市占率排列）， 所以下面鸟哥就针对这三家的显卡驱动程序安装，作个简单的介绍吧！

由于硬件驱动程序与核心有关，因此你想要安装这个驱动程序之前，请务必先参考第二十一章与第二十二章的介绍，才能够顺利的编译出显卡驱动程序喔！ 建议可以直接使用 yum 去安装“ Development Tools ”这个软件群组以及 kernel-devel 这个软件即可。



Tips 因为你得要有实际的硬件才办法安装这些驱动程序，因此下面鸟哥使用的则是实体机器上面装有个别的显卡的设备，就不是使用虚拟机了喔！

23.3.1 NVidia

虽然 Xorg 已经针对 NVidia 公司的显卡驱动程序提供了 "nouveau" 这个模块，不过这个模块无法提供很多额外的功能。 因此，如果你想要使用新的显卡功能时，就得要额外安装 NVidia 提供的给 Linux 的驱动程序才行。

至于 NVidai 虽然有提供驱动程序给大家使用，不过他们并没有完全释出，因此自由软件圈不能直接拿人家的东西来重新开发！ 不过还是有很多好心人士有提供相关的软件库给大家使用啦！你可以自行 google 查阅相关的软件库 （比较可惜的是，EPEL 里面并没有 NVidia 官网释出的驱动程序就是了！）所以，下面我们还是使用传统的从 NVidia 官网上面下载相关的软件来安装的方式喔！

- 查询硬件与下载驱动程序

你得要先确认你的硬件为何才可以下载到正确的驱动程序啊！简单查询的方法可以使用 lspci 喔！还不需要拆主机机箱啦！

```
[root@study ~]# lspci &#124; grep -Ei '（vga&#124;display）'
00:02.0 Display controller: Intel Corporation Xeon E3-1200 v3/4th Gen Core Processor Inte
        Graphics Controller （rev 06）
01:00.0 VGA compatible controller: NVIDIA Corporation GF119 [GeForce GT 610] （rev a1）
# 鸟哥选的这部实体机器测试中，其实有内置 Intel 显卡以及 NVidia GeForece GT610 这两张卡！
# 屏幕则是接在 NVidia 显卡上面喔！
```

建议你可以到 NVidia 的官网 （http://www.nvidia.com.tw） 自行去下载最新的驱动程序，你也可以到下面的链接直接查阅给 Linux 用的驱动程序：

- http://www.nvidia.com.tw/object/unix_tw.html

请自行选择与你的系统相关的环境。现在 CentOS 7 都仅有 64 位啊！所以不要怀疑，就是选择 Linux x86_64/AMD64/EM64T 的版本就对了！ 不过还是得要注意你的 GPU 是旧的还是新的喔～像鸟哥刚刚查到上面使用的是 GT610 的显卡，那使用最新长期稳定版就可以了！ 鸟哥下载的版本文件名有点像： NVIDIA-Linux-x86_64-352.41.run，我将这文件名放置在 /root 下面喔！接下来就是这样作：

- 系统升级与取消 nouveau 模块的载入

因为这部系统是新安装的，所以没有我们虚拟机里面已经安装好所有需要的环境了。因此，我们建议你最好是做好系统升级的动作， 然后安装所需要的编译环境，最后还得要将 nouveau 模块排除使用！因为强迫系统不要使用 nouveau 这个驱动，这样才能够完整的让 nvidia 的驱动程序运行！ 那就来瞧瞧怎么作啰！

```
# 1\. 先来全系统升级与安装所需要的编译程序与环境；
[root@study ~]# yum update
[root@study ~]# yum groupinstall "Development Tools"
[root@study ~]# yum install kernel-devel kernel-headers

# 2\. 开始处理不许载入 nouveau 模块的动作！
[root@study ~]# vim /etc/modprobe.d/blacklist.conf  # 这文件默认应该不存在
blacklist nouveau
options nouveau modeset=0

[root@study ~]# vim /etc/default/grub
GRUB_CMDLINE_LINUX="vconsole.keymap=us crashkernel=auto  vconsole.font=latarcyrheb-sun16
  rhgb quiet rd.driver.blacklist=nouveau nouveau.modeset=0"
# 在 GRUB_CMDLINE_LINUX 设置里面加上 rd.driver.blacklist=nouveau nouveau.modeset=0 的意思！
[root@study ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
[root@study ~]# reboot

[root@study ~]# lsmod &#124; grep nouveau
# 最后要没有出现任何模块才是对的！
```

- 安装驱动程序

要完成上述的动作之后才能够处理下面的行为喔！（文件名依照你的环境去下载与执行）：

```
[root@study ~]# systemctl isolate multi-user.target
[root@study ~]# sh NVIDIA-Linux-x86_64-352.41.run
# 接下来会出现下面的数据，请自行参阅图示内容处理啰！
```



图 23.3.1-1、Nvidia 官网驱动程序相关设置画面示意

上面说的是授权，你必须要接受 （Accept） 才能继续。

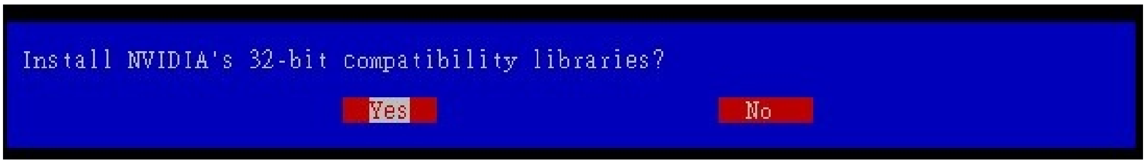


图 23.3.1-2、Nvidia 官网驱动程序相关设置画面示意

要不要安装 32 位相容的函数库，鸟哥个人是认为还是装一下比较好啦！



图 23.3.1-3、Nvidia 官网驱动程序相关设置画面示意

让这支安装程序主动的去修改 xorg.conf 吧！比较轻松愉快！就按下 Yes 即可。

最后按下 OK 就结束安装啰！这个时候如果你去查阅一下 /etc/X11/xorg.conf 的内容，会发现 Device 的 Driver 设置会成为 nvidia 喔！这样就搞定啰！很简单吧！ 而且这个时候你的 /usr/lib64/xorg/modules/drivers 目录内，会多出一个 nvidia_drv.so 的驱动程序文件啰！ 同时这个软件还提供了一支很有用的程序来帮助我们进行驱动程序升级喔！

```
[root@study ~]# nvidia-installer --update
# 可以进行驱动程序的升级检查喔！
```

好啰，那你就赶紧试看看新的显卡芯片的功能吧。而如果有什么疑问的话，查阅一下 /var/log/nvidia* 开头的登录文件看看吧！ ^_^

23.3.2 AMD （ATI）

AMD 的显卡 （ATI） 型号也很多，不过因为 AMD 的显卡有提供成为 Open Source ，目前有个名为 ELrepo 的网站有主动提供 AMD 的显卡驱动喔！ 而且是针对我们 CentOS 7 耶～好像还不赖～其实 ELrepo 也提供了 NVidia 的驱动程序啦！只是型号太多，所以鸟哥还是使用 NVidia 官网的数据来教学而已。

那如何取得 ELrepo 呢？这个网站主文件夹在下面，你可以自己瞧一瞧，至于安装 ELrepo 的 yum 配置文件方式如下：

- http://elrepo.org

```
[root@study ~]# rpm --import https://www.elrepo.org/RPM-GPG-KEY-elrepo.org
[root@study ~]# rpm -Uvh http://www.elrepo.org/elrepo-release-7.0-2.el7.elrepo.noarch.rpm

[root@study ~]# yum clean all
[root@study ~]# yum --enablerepo elrepo-testing search fglrx
kmod-fglrx.x86_64 : fglrx kernel module（s）
fglrx-x11-drv.x86_64 : AMD's proprietary driver for ATI graphic cards # 这就对了！
fglrx-x11-drv-32bit.x86_64 : Compatibility 32-bit files for the 64-bit Proprietary AMD dr
fglrx-x11-drv-devel.x86_64 : Development files for AMD OpenGL X11 display driver.

[root@study ~]# yum --enablerepo elrepo-testing install fglrx-x11-drv
# 很快的！这样就安装好了 AMD 的显卡驱动程序了耶！超开心的吧！
```

安装完毕后，系统就会在 /usr/lib64/xorg/modules/drivers/ 里面出现 fglrx_drv.so 这个新的驱动程序啦！ 与 Nvidia 相同的， ATI 也提供一支名为 aticonfig 的指令来帮忙设置 xorg.conf ，你可以直接输入“ aticonfig -v ”来看看处理的方式即可。然后你就可以重新启动 X 来看看新的驱动程序功能啰！非常简单吧！

23.3.3 Intel

老实说，由于 Intel 针对 Linux 的图形接口驱动程序已经开放成为 Open source 了，所以理论上你不需要重新安装 Intel 的显卡驱动程序的。除非你想要使用比默认的更新的驱动程序，那么才需要重新安装下面的驱动程序。 Intel 对 Linux 的显卡驱动程序已经有独立的网站在运行，如下的链接就是安装的说明网页：

- https://01.org/zh/linuxgraphics

其实 Intel 的显卡用的地方非常的多喔！因为只要是整合型主板芯片组，用的是 Intel 的芯片时， 通常都整合了 Intel 的显卡啰～鸟哥使用的一组 cluster 用的就是 Intel 的芯片，所以啰～这家伙也是用的到的啦！

一般来说， Intel 的显卡都常常会使用 i910 等驱动程序，而不是这个较新的 intel 驱动程序！ 你可以察看一下你系统是否有存在这些文件：

```
[root@study ~]# locate libdrm
/usr/lib64/libdrm.so.2
/usr/lib64/libdrm.so.2.4.0
/usr/lib64/libdrm_intel.so.1      # 就是这几个怪东西！
/usr/lib64/libdrm_intel.so.1.0.0
.....（下面省略）.....

[root@study ~]# locate intel &#124; grep xorg
/usr/lib64/xorg/modules/drivers/intel_drv.so
# 上面这个就是 Intel 的显卡驱动程序了！
```

呼呼！我们的 CentOS 有提供新的 Intel 显卡驱动程序啦！所以不需要重新安装说～ 只是可能需要修改 xorg.conf 这个配置文件的内容。基本上，要修改的地方有：

```
[root@study ~]# vi /etc/X11/xorg.conf
Section "Device"
        Identifier  "Videocard0"
        Driver      "intel"  &lt;==原本可能会是使用 i91x 喔
EndSection

Section "Module"
        ....（中间省略）....
        Load  "glx"    &lt;==这两个很重要！务必要载入！
        Load  "dri"
        ....（中间省略）....
EndSection

Section "DRI"          &lt;==这三行是新增的！让大家都能使用 DRI
       Mode 0666       &lt;==基本上，就是权限的设置
EndSection
```

如果一切顺利的话，接下来就是重新启动 X 啰～使用新的 Intel 驱动程序吧！加油啰！



Tips 老实说，CentOS 7 的 Xorg 自动侦测程序作的其实还不错，在鸟哥这次测试实体机器的系统上面安装的图形界面时，几乎 Xorg 都可以正确的抓到驱动程序， 连双屏幕功能也都可以顺利的启用没问题。所以除非必要，否则您应该不需要重新设置 xorg.conf 喔！ ^_^

23.4 重点回顾

- Unix Like 操作系统上面的 GUI 使用的是最初由 MIT 所开发的 X window system，在 1987 释出 X11 版，并于 1994 更改为 X11R6 ，故此 GUI 接口也被称为 X 或 X11
- X window system 的 X server 最初由 XFree86 计划所开发，后来则由 Xorg 基金会所持续开发；
- X window system 主要分为 X server 与 X client ，其中 X Server 在管理硬件，而 X Client 则是应用程序。
- 在运行上，X Client 应用程序会将所想要呈现的画面告知 X Server ，最终由 X server 来将结果通过他所管理的硬件绘制出来！
- 每一支 X client 都不知道对方的存在，必须要通过特殊的 X client ，称为 Window Manager 的， 来管理各窗口的重叠、移动、最小化等工作。
- 若有需要登陆图形接口，有时会有 Display Manager 来管理这方面的动作
- startx 可以侦测 X server / X client 的启动脚本，并调用 xinit 来分别执行；
- X 可以启动多个，各个 X 显示的位置使用 -display 来处理，显示位置为 :0, :1...
- Xorg 是一个 X server ，配置文件位于 /etc/X11/xorg.conf ，里面含有 Module, Files, Monitor, Device 等设置阶段。目前较新的设置中， 会将额外的设置放置于 /etc/X11/xorg.conf.d/*.conf

23.5 本章习题

（ 要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看 ）

- 在 X 设置没问题的情况下，你在 Linux 主机如何取得窗口接口？如果是在 multi-user.target 模式下，可以使用 startx 进入，至于 graphical.target ，则直接进入 tty1 即可使用 display manager 登陆 X Window 系统。
- 利用 startx 可以在 multi-user.target 的环境下进入 X Window 系统。请问 startx 的主要功能？整个 X 窗口系统的重点在启动 X server 并载入 X client ，而执行 X server/X client 调用的任务为 xinit ，startx 只是一个较为友好的脚本程序，可以搜寻系统上面的 X server / X client 设置值， 以提供 xinit 来执行而已。
- 如何知道你系统当中 X 系统的版本与计划？最简单可以利用 root 的身份下达 X -version 或 Xorg -version 即可知道！
- 要了解为何 X 系统可以允许不同硬件、主机、操作系统之间的沟通，需要知道 X server / X client 的相关知识。 请问 X Server / X client / Window manager 的主要用途功能？X Server 主要负责屏幕的绘制，以及周边输入设备如鼠标、键盘等数据的收集，并回报给 X Client ； X Client 主要负责数据的运算，收到来自 X Server 的数据后，加以运算得到图形的数据，并回传给 X Server， 让 X server 自行绘制图形。至于 Window manager 是一个比较特殊的 X Client ，他可以管理更多控制元素， 最重要的地方还是在于窗口的大小、重叠、移动等等的功能。
- 如何重新启动 X
    - 最简单在 X Window System 下，直接按下 [alt]+[ctrl]+[backspace<--] 即可
    - 也可以 systemctl isolate multi-usertarget 再 systemctl isolate graphical.target
    - 也可以关闭 X 后，再 startx 启动等等。
- 试说明 ~/.xinitrc 这个文件的用途？当我们要启动 X 时，必须要启动 X Client 软件端。这个 ~/.xinitrc 即是在客制化自己的 X Client ， 你可以在这个文件内输入你自己的 X Client 。若无此文件，则默认以 /etc/X11/xinit/xinitrc 替代。
- 我在 CentOS 的系统中，默认使用 GNOME 登陆 X 。但我想要改以 KDE 登陆，该怎么办？
    - 首先你必须要已经安装 KDE 环境 （参考前一章的 yum grouplist 功能），
    - 然后可以借由修改 /etc/sysconfig/desktop 内的设置值即可。
    - 但如果你不是 root 无法修订该文件时，亦可以在自己的主文件夹参考 /etc/X11/xinit/xinitrc 的内容自行制作 ~/.xinitrc 文件来修改！
- X Server 的 port 默认开放在？目前默认并不会启动 TCP 端口。不过如果经过设置，则 X port 默认开放在 port 6000 ，而且称此一显示为 :0
- Linux 主机是否可以有两个以上的 X是的！可以！第一个 X 通常在 tty1 ，第二个在 tty2 以后，依序类推。 第几个是以启动的顺序来定义，并非 :0 , :1 的意思～

- X Server 的配置文件是 xorg.conf，在该文件中， Section Files 干嘛用的？相当重要！是设置显示字体用的。而字体一般放置目录在 /usr/share/X11/fonts/ 及 /usr/share/fonts/ 当中。
- 我发现我的 X 系统键盘所输入的字母老是打不出我所需要的单字，可能原因该如何修订？应该是键盘符号对应表跑掉了。可以修改 xorg.conf 文件内， 关于 Keyboard 的 Option XkbLayout 项目，将他改为 us 即可！
- 当我的系统内有安装 GNOME 及 KDE 两个 X Widnow Manager ，我原本是以 KDE 为默认的 WM， 若想改为 GNOME 时，应该如何修改？修改 /etc/sysconfig/desktop 内部，成为 GNOME 即可！

23.6 参考资料与延伸阅读

- [1]维基百科对 X Window 的介绍：http://en.wikipedia.org/wiki/X_Window_System
- [2]X Server/X client 与网络相关性的参考图示：http://en.wikipedia.org/wiki/File:X_client_sever_example.svg
- [3]系统的 man page： man xinit 、 man Xorg 、 man startx
- [4]一些与中文字体有关的网页链接： 洪朝贵老师评论员的字体设置：http://www.cyut.edu.tw/~ckhung/b/gnu/font.php
- X 相关的官方网站： X.org 官方网站 （http://www.x.org/）、 XFree86 官方网站（http://www.xfree86.org/）

2003/02/12：第一次完成 2005/06/29：将旧的文章移动到 这里 。如果你需要旧版的 xf86config 与相关的工具，则请前往该旧文章查阅！ 2005/07/11：经历了许多的时间，将主机的配置文件重复改了改，终于完成一些简单的 X 测试！ 2006/11/07：经由网友x1215 这一篇的介绍，得知该网站，赶紧去处理！ 2009/07/03：将旧版基于 FC4 的版本移动到此处 2009/07/15：奋战好几天，将驱动程序安装加上，同时加入字体管理功能。 2009/07/28：网友 LazyBug Chan 兄热情回报，使用 XFCE 的 Ubuntu 是 Xubuntu 这个分支！感谢回报！ 2009/08/07：加入 Window Manger 的全名与链接 2015/09/11：将旧的基于 CentOS 5.x 的版本移动到 此处

第二十四章、Linux 核心编译与管理

最近更新日期：20//

我们说的 Linux 其实指的就是核心 （kernel） 而已。这个核心控制你主机的所有硬件并提供系统所有的功能， 所以说，他重不重要啊！我们开机的时候其实就是利用开机管理程序载入这个核心文件来侦测硬件， 在核心载入适当的驱动程序后，你的系统才能够顺利的运行。现今的系统由于强调线上升级机制，因此非常不建议自订核心编译！ 但是，如果你想要将你的 Linux 安装到 U盘、想要将你的 Eee PC 小笔记本安装自己的 Linux ， 想让你的 Linux 可以驱动你的小家电，此时，核心编译就是相当重要的一个任务了！ 这一篇比较进阶，如果你对系统移植没有兴趣的话，这一篇可以先略过喔！ ^_^

24.1 编译前的任务：认识核心与取得核心源代码

我们在第一章里面就谈过 Linux 其实指的是核心！这个“核心 （kernel）”是整个操作系统的最底层，他负责了整个硬件的驱动，以及提供各种系统所需的核心功能，包括防火墙机制、是否支持 LVM 或 Quota 等文件系统等等，这些都是核心所负责的！所以啰，在第十九章的开机流程中，我们也会看到 MBR 内的 loader 载入核心文件来驱动整个系统的硬件呢！ 也就是说，如果你的核心不认识某个最新的硬件，那么该硬件也就无法被驱动，你当然也就无法使用该硬件啰！

24.1.1 什么是核心 （Kernel）

这已经是整个 Linux 基础的最后一篇了，所以，下面这些数据你应该都要“很有概念”才行～ 不能只是“好像有印象”～好了，那就复习一下核心的相关知识吧！

- Kernel

还记得我们在第十章的 BASH shell 提到过：计算机真正在工作的东西其实是“硬件”， 例如数值运算要使用到 CPU、数据储存要使用到硬盘、图形显示会用到显卡、音乐发声要有音效芯片、连接 Internet 可能需要网卡等等。那么如何控制这些硬件呢？那就是核心的工作了！也就是说，你所希望计算机帮你达成的各项工作， 都需要通过“核心”的帮助才行！当然啰，如果你想要达成的工作是核心所没有提供的， 那么你自然就没有办法通过核心来控制计算机使他工作啰！

举例来说，如果你想要有某个网络功能 （例如核心防火墙机制） ，但是你的核心偏偏忘记加进去这项功能， 那么不论你如何“卖力”的设置该网络套件，很抱歉！不来电！换句话说，你想要让计算机进行的工作，都必须要“核心有支持”才可以！这个标准不论在 Windows 或 Linux 这几个操作系统上都相同！如果有一个人开发出来一个“全新的硬件”，目前的核心不论 Windows 或 Linux 都不支持，那么不论你用什么系统，哈哈！这个硬件都是英雄无用武之地啦！ 那么是否了解了“核心”的重要了呢？所以我们才需要来了解一下如何编译我们的核心啦！

那么核心到底是什么啊？其实核心就是系统上面的一个文件而已， 这个文件包含了驱动主机各项硬件的侦测程序与驱动模块。在第十九章的开机流程分析中，我们也提到这个文件被读入内存的时机， 当系统读完 BIOS 并载入 MBR 内的开机管理程序后，就能够载入核心到内存当中。然后核心开始侦测硬件， 挂载根目录并取得核心模块来驱动所有的硬件，之后调用 systemd 就能够依序启动所有系统所需要的服务了！

这个核心文件通常被放置成 /boot/vmlinuz-xxx ，不过也不见得， 因为一部主机上面可以拥有多个核心文件，只是开机的时候仅能选择一个来载入而已。 甚至我们也可以在一个 distribution 上面放置多个核心，然后以这些核心来做成多重开机呢！

- 核心模块 （kernel module） 的用途

既然核心文件都已经包含了硬件侦测与驱动模块，那么什么是核心模块啊？要注意的是，现在的硬件更新速度太快了，如果我的核心比较旧，但我换了新的硬件，那么，这个核心肯定无法支持！怎么办？重新拿一个新的核心来处理吗？开玩笑～核心的编译过程可是很麻烦的～

所以啰，为了这个缘故，我们的 Linux 很早之前就已经开始使用所谓的模块化设置了！亦即是将一些不常用的类似驱动程序的咚咚独立出核心，编译成为模块，然后，核心可以在系统正常运行的过程当中载入这个模块到核心的支持。如此一来，我在不需要更动核心的前提之下，只要编译出适当的核心模块，并且载入他，呵呵！我的 Linux 就可以使用这个硬件啦！简单又方便！

那我的模块放在哪里啊？可恶！怎么会问这个傻问题呢？当然一定要知道的啦！就是 /lib/modules/$（uname -r）/kernel/ 当中啦！

- 自制核心 - 核心编译

刚刚上面谈到的核心其实是一个文件，那么这个文件怎么来的？当然是通过源代码（source code）编译而成的啊！因为核心是直接被读入到内存当中的，所以当然要将他编译成为系统可以认识的数据才行！也就是说，我们必须要取得核心的源代码，然后利用第二十一章 Tarball 安装方式提到的编译概念来达成核心的编译才行啊！（这也是本章的重点啊！^_^）

- 关于驱动程序 - 是厂商的责任还是核心的责任？

现在我们知道硬件的驱动程序可以编译成为核心模块，所以可以在不改变核心的前提下驱动你的新硬件。但是，很多朋友还是常常感到困惑，就是 Linux 上面针对最新硬件的驱动程序总是慢了几个脚步，所以觉得好像 Linux 的支持度不足！其实不可以这么说的，为什么呢？因为在 Windows 上面，对于最新硬件的驱动程序需求，基本上，也都是厂商提供的驱动程序才能让该硬件工作的，因此，在这个“驱动程序开发”的工作上面来说，应该是属于硬件发展厂商的问题，因为他要我们买他的硬件，自然就要提供消费者能够使用的驱动程序啦！

所以，如果大家想要让某个硬件能够在 Linux 上面跑的话，那么似乎可以发起一人一信的方式，强烈要求硬件开发商发展 Linux 上面的驱动程序！这样一来，也可以促进 Linux 的发展呢！

24.1.2 更新核心的目的

除了 BIOS（或 UEFI）之外，核心是操作系统中最早被载入到内存的咚咚，他包含了所有可以让硬件与软件工作的信息，所以，如果没有搞定核心的话，那么你的系统肯定会有点小问题！好了，那么是不是将“所有目前核心有支持的东西都给他编译进去我的核心中，那就可以支持目前所有的硬件与可执行的工作啦！”！

这话说的是没错啦，但是你是否曾经看过一个为了怕自己今天出门会口渴、会饿、会冷、会热、会被车撞、会摔跤、会被性骚扰，而在自己的大包包里面放了大瓶矿泉水、便当、厚外套、短裤、防撞钢梁、止滑埝、电击棒....等一大堆东西，结果却累死在半路上的案例吗？当

然有！但是很少啦！我相信不太有人会这样做！ （会这么做的人通常都已经在医院了～） 取而代之的是会看一下天气，冷了就只带外套， 热了就只带短衣、如果穿的漂亮一点又预计晚点回家就多带个电击棒、 出远门到没有便利商店的地方才多带矿泉水....

说这个干什么！对啦！就是要你了解到，核心的编译重点在于“你要你的 Linux 作什么？”，是啦！如果没有必要的工作，就干脆不要加在你的核心当中了！这样才能让你的 Linux 跑得更稳、更顺畅！这也是为什么我们要编译核心的最主要原因了！

- Linux 核心特色，与默认核心对终端用户的角色

Linux 的核心有几个主要的特色，除了“Kernel 可以随时、随各人喜好而更动”之外，Kernel 的“版本更动次数太频繁”也是一个特点！所以啰，除非你有特殊需求， 否则一次编译成功就可以啦！不需要随时保持最新的核心版本，而且也没有必要 （编译一次核心要粉久的了ㄟ！） 。

那么是否“我就一定需要在安装好了 Linux 之后就赶紧给他编译核心呢？”， 老实说，“并不需要的”！这是因为几乎每一个 distribution 都已经默认编译好了相当大量的模块了， 所以使用者常常或者可能会使用到的数据都已经被编译成为模块，也因此，呵呵！ 我们使用者确实不太需要重新来编译核心！尤其是“一般的使用者， 由于系统已经将核心编译的相当的适合一般使用者使用了，因此一般入门的使用者，基本上， 不太需要编译核心”。

- 核心编译的可能目的

OK！那么鸟哥闲闲没事干跑来写个什么东西？既然都不需要编译核心还写编译核心的分享文章， 鸟哥卖弄才学呀？很抱歉，鸟哥虽然是个“不学有术”的混混，却也不会平白无故的写东西请您来指教～ 当然是有需要才会来编译核心啦！编译核心的时机可以归纳为几大类：

- 新功能的需求： 我需要新的功能，而这个功能只有在新的核心里面才有，那么为了获得这个功能，只好来重新编译我的核心了。例如 iptables 这个防火墙机制只有在 2.4.xx 以后的版本里面才有，而新开发的主板芯片组， 很多也需要新的核心推出之后，才能正常而且有效率的工作！
- 原本核心太过臃肿： 如果你是那种对于系统“稳定性”很要求的人，对于核心多编译了很多莫名其妙的功能而不太喜欢的时候， 那么就可以重新编译核心来取消掉该功能啰；
- 与硬件搭配的稳定性： 由于原本 Linux 核心大多是针对 Intel 的 CPU 来作开发的，所以如果你的 CPU 是 AMD 的系统时，有可能 （注意！只是有可能，不见得一定会如此） 会让系统跑得“不太稳！”。此外，核心也可能没有正确的驱动新的硬件，此时就得重新编译核心来让系统取得正确的模块才好。
- 其他需求 （如嵌入式系统） ：就是你需要特殊的环境需求时，就得自行设计你的核心啰！（ 像是一些商业的套装软件系统，由于需要较为小而美的操作系统， 那么他们的核心就需要更简洁有力了！）



Tips 话说，2014 年鸟哥为了要搞定 banana pi （一种单版计算机，或者可以称为手机的硬件拿来作 Linux 安装的硬件） 的 CPU 最高频率限制， 因为该限制是直接写入到 Linux 核心当中的，这时就只好针对该硬件的 Linux 核心，修改不到 10 行的程序码之后，重新编译！ 才能将原本限制到 900MHz 的频率提升到 1.2GHz 哩！

另外，需要注意重新编译核心虽然可以针对你的硬件作最优化的步骤 （例如刚刚提到的 CPU 的问题！） ，不过由于这些最优化的步骤对于整体性能的影响是很小很小的， 因此如果是为了增加性能来编译核心的话，基本上，效益不大！然而，如果是针对“系统稳定性”来考虑的话， 那么就有充分的理由来支持你重新编译核心啰！

“如果系统已经运行很久了，而且也没有什么大问题， 加上我又不增加冷门的硬件设备，那么建议就不需要重新编译核心了”， 因为重新编译核心的最主要目的是“想让系统变的更稳！”既然你的 Linux 主机已经达到这个目的了，何必再编译核心？不过，就如同前面提到的， 由于默认的核心不见得适合你的需要，加上默认的核心可能并无法与你的硬件配备相配合， 此时才开始考虑重新编译核心吧！



Tips 早期鸟哥是强调最好重新编译核心的一群啦！不过，这个想法改变好久了～ 既然原本的 distribution 都已经帮我们考虑好如何使用核心了，那么， 我们也不需要再重新的编译核心啦！尤其是 distribution 都会主动的释出新版的核心 RPM 版本， 所以，实在不需要自己重新编译的！当然啦，如同前面提到的，如果你有特殊需求的话，那就另当别论噜！ ^_^

由于“核心的主要工作是在控制硬件！”所以编译核心之前， 请先了解一下你的硬件配备，与你这部主机的未来功能！由于核心是“越简单越好！”所以只要将这部主机的未来功能给他编进去就好了！ 其他的就不用去理他啦！

24.1.3 核心的版本

核心的版本问题，我们在第一章已经谈论过， 目前 CentOS 7 使用的 3.10.x 版本为长期维护版本，不过理论上我们也可以升级到后续的主线版本上面！不会像以前 2.6.x 只能升级到 2.6.x 的后续版本， 而不能改成其他主线版本。不过这也只是“理论上”而已，因为目前许多的软件依旧与核心版本有关，例如那个虚拟化软件 qemu 之类的， 与核心版本之间是有搭配性的关系的，所以，除非你要一口气连同核心相依的软件通通升级，否则最好使用长期维护版本的最新版来处理较佳。

举例来说，CentOS 7 使用的是 3.10.0 这个长期版本，而目前（2015/09）这个 3.10 长期版本，最新的版本为 3.10.89，意思是说，你最好是拿 3.10.89 来作为核心升级的依据，而不是拿最新的 4.2.1 来升级的意思。

虽然理论上还是拿自家长期维护版本的最新版本来处理比较好，不过鸟哥因为需要研究虚拟化的 PCI passthrough 技术，确实也曾经在 CentOS 7.1 的系统中将 3.10.x 的版本升级到 4.2.3 这个版本上！这样才完成了 VGA 的 PCI passthrough 功能！所以说，如果你真的想要使用较新的版本来升级，也不是不可以，只是后果会发生什么问题，就得要自行负责啰！

24.1.4 核心源代码的取得方式

既然核心是个文件，要制作这个文件给系统使用则需要编译，既然要有编译，当然就得要有源代码啊！那么源代码怎么来？基本上，依据你的 distributions 去挑选的核心源代码来源主要有：

- 原本 distribution 提供的核心源代码文件

事实上，各主要 distributions 在推出他们的产品时，其实已经都附上了核心源代码了！不过因为目前数据量太庞大，因此 SRPM 默认已经不给映射站下载了！主要的源代码都放置于下面的网站上：

- 全部的 CentOS 原始 SRPM：http://vault.centos.org/
- CentOS 7.1 的 SRPM：http://vault.centos.org/7.1.1503/

CentOS 7.x 开始的版本中，其版本后面会接上释出的日期，因为 CentOS 7.1 是 2015/03 释出的，因此它的下载点就会是在 7.1.1503 啰！1503 指的就是 2015/03 的意思～你可以进入上述的网站后，到 updates 目录下，一层一层的往下找，就可以找到 kernel 相关的 SRPM 啰！

你或许会说：既然要重新编译，那么干嘛还要使用原本 distributions 释出的源代码啊？真没创意～话不是这么说，因为原本的 distribution 释出的源代码当中，含有他们设置好的默认设置值，所以，我们可以轻易的就了解到当初他们是如何选择与核心及模块有关的各项设置项目的参数值，那么就可以利用这些可以配合我们 Linux 系统的默认参数来加以修改，如此一来，我们就可以“修改核心，调整到自己喜欢的样子”啰！而且编译的难度也会比较低一点！

- 取得最新的稳定版核心源代码

虽然使用 distribution 释出的核心 source code 来重新编译比较方便，但是，如此一来，新硬件所需要的新驱动程序，也就无法借由原本的核心源代码来编译啊！所以啰，如果是站在要更新驱动程序的立场来看，当然使用最新的核心可能会比较好啊！

Linux 的核心目前是由其发明者 Linus Torvalds 所属团队在负责维护的，而其网站在下面的站址上，在该网站上可以找到最新的 kernel 信息！不过，美中不足的是目前的核心越来越大了（linux-3.10.89.tar.gz 这一版，这一个文件大约 105MB 了！），所以如果你的 ISP 连外很慢的话，那么使用台湾的映射站台来下载不失为一个好方法：

- 核心官网：http://www.kernel.org/
- 交大资料：ftp://linux.cis.nctu.edu.tw/kernel/linux/kernel/
- 国高中心：ftp://ftp.twaren.net/pub/Unix/Kernel/linux/kernel/

- 保留原本设置：利用 patch 升级核心源代码

如果 （1）你曾经自行编译过核心，那么你的系统当中应该已经存在前几个版本的核心源代码， 以及上次你自行编译的参数设置值才对； （2）如果你只是想要在原本的核心下面加入某些特殊功能， 而该功能已经针对核心源代码推出 patch 补丁文件时。那你该如何进行核心源代码的更新，以便后续的编译呢？

其实每一次核心释出时，除了释出完整的核心压缩文件之外，也会释出“该版本与前一版本的差异性 patch 文件”， 关于 patch 的制作我们已经在第二十一章当中提及， 你可以自行前往参考。这里仅是要提供给你的信息是，每个核心的 patch 仅有针对前一版的核心来分析而已，所以，万一你想要由 3.10.85 升级到 3.10.89 的话，那么你就得要下载 patch-3.10.86, patch-3.10.87, patch-3.10.88, patch-3.10.89 等文件，然后“依序”一个一个的去进行 patch 的动作后， 才能够升级到 3.10.89 喔！这个重要！不要忘记了。

同样的，如果是某个硬件或某些非官方认定的核心添加功能网站所推出的 patch 文件时，你也必须要了解该 patch 文件所适用的核心版本，然后才能够进行 patch ，否则容易出现重大错误喔！这个项目对于某些商业公司的工程师来说是很重要的。 举例来说，鸟哥的一个高中同学在业界服务，他主要是进行类似 Eee PC 开发的计划，然而该计划的硬件是该公司自行推出的！ 因此，该公司必须要自行搭配核心版本来设计他们自己的驱动程序，而该驱动程序并非 GPL 授权，因此他们就得要自行将驱动程序整合进核心！如果改天他们要将这个驱动程序释出，那么就得要利用 patch 的方式， 将硬件驱动程序文件释出，我们就得要自行以 patch 来更新核心啦！

在进行完 patch 之后，你可以直接检查一下原本的设置值，如果没有问题， 就可以直接编译，而不需要再重新的选择核心的参数值，这也是一个省时间的方法啊！ 至于 patch file 的下载，同样是在 kernel 的相同目录下，寻找文件名是 patch 开头的就是了。

24.1.5 核心源代码的解压缩/安装/观察

其实，不论是从 CentOS 官网取得的 SRPM 或者是从 Linux kernel 官网取得的 tarball 核心源代码，最终都会有一个 tarball 的核心源代码就是了！ 因此，鸟哥从 linux kernel 官网取得 linux-3.10.89.tar.xz 这个核心文件，这个核心文件的源代码是从下面的网址取得的：

- ftp://ftp.twaren.net/pub/Unix/Kernel/linux/kernel/v3.x/linux-3.10.89.tar.xz

- 核心源代码的解压缩与放置目录

鸟哥这里假设你也是下载上述的链接内的文件，然后该文件放置到 /root 下面。由于 Linux 核心源代码一般建议放置于 /usr/src/kernels/ 目录下面，因此你可以这样处理：

```
[root@study ~]# tar -Jxvf linux-3.10.89.tar.xz -C /usr/src/kernels/
```

此时会在 /usr/src/kernels 下面产生一个新的目录，那就是 linux-3.10.89 这个目录啰！ 我们在下个小节会谈到的各项编译与设置，都必须要在这个目录下面进行才行喔！好了，那么这个目录下面的相关文件有啥咚咚？ 下面就来谈谈：

- 核心源代码下的次目录

在上述核心目录下含有哪些重要数据呢？基本上有下面这些东西：

- arch ：与硬件平台有关的项目，大部分指的是 CPU 的类别，例如 x86, x86_64, Xen 虚拟支持等；
- block ：与区块设备较相关的设置数据，区块数据通常指的是大量储存媒体！还包括类似 ext3 等文件系统的支持是否允许等。
- crypto ：核心所支持的加密的技术，例如 md5 或者是 des 等等；
- Documentation ：与核心有关的一堆说明文档，若对核心有极大的兴趣，要瞧瞧这里！
- drivers ：一些硬件的驱动程序，例如显卡、网卡、PCI 相关硬件等等；
- firmware ：一些旧式硬件的微指令码 （固件） 数据；
- fs ：核心所支持的 filesystems ，例如 vfat, reiserfs, nfs 等等；
- include ：一些可让其他程序调用的标头 （header） 定义数据；
- init ：一些核心初始化的定义功能，包括挂载与 init 程序的调用等；
- ipc ：定义 Linux 操作系统内各程序的沟通；
- kernel ：定义核心的程序、核心状态、线程、程序的调度 （schedule） 、程序的讯号 （signle） 等
- lib ：一些函数库；
- mm ：与内存单元有关的各项数据，包括 swap 与虚拟内存等；
- net ：与网络有关的各项协定数据，还有防火墙模块 （net/ipv4/netfilter/*） 等等；
- security ：包括 selinux 等在内的安全性设置；
- sound ：与音效有关的各项模块；
- virt ：与虚拟化机器有关的信息，目前核心支持的是 KVM （Kernel base Virtual Machine）

这些数据先大致有个印象即可，至少未来如果你想要使用 patch 的方法加入额外的新功能时， 你要将你的源代码放置于何处？这里就能够提供一些指引了。当然，最好还是跑到 Documentation 那个目录下面去瞧瞧正确的说明， 对你的核心编译会更有帮助喔！

24.2 核心编译的前处理与核心功能选择

什么？核心编译还要进行前处理？没错啦！事实上，核心的目的在管理硬件与提供系统核心功能，因此你必须要先找到你的系统硬件， 并且规划你的主机未来的任务，这样才能够编译出适合你这部主机的核心！所以，整个核心编译的重要工作就是在“挑选你想要的功能”。 下面鸟哥就以自己的一部主机软/硬件环境来说明，解释一下如何处理核心编译啰！

24.2.1 硬件环境检视与核心功能要求

鸟哥的一部主机硬件环境如下 （在虚拟机中，通过 /proc/cpuinfo 及 lspci 观察）：

- CPU：Intel（R） Xeon（R） CPU E5-2650
- 主板芯片组： KVM 虚拟化仿真的主版 （Intel 440FX 相容）
- 显卡： Red Hat, Inc. QXL paravirtual graphic card
- 内存： 2.0GB 内存
- 硬盘： KVM Virtio 界面磁盘 40G （非 IDE/SATA/SAS 喔！）
- 网卡： Red Hat, Inc Virtio network device

硬件大致如上，至于这部主机的需求，是希望做为未来在鸟哥上课时，可以通过虚拟化功能来处理学生的练习用虚拟机。 这部主机也是鸟哥用来放置学校上课教材的机器，因此，这部主机的 I/O 需求须要好一点，未来还需要打开防火墙、 WWW 服务器功能、FTP 服务器功能等，基本上，用途就是一部小型的服务器环境啰。大致上需要这样的功能啦！

24.2.2 保持干净源代码： make mrproper

了解了硬件相关的数据后，我们还得要处理一下核心源代码下面的残留文件才行！假设我们是第一次编译， 但是我们不清楚到下面载下来的源代码当中有没有保留目标文件 （*.o） 以及相关的配置文件存在， 此时我们可以通过下面的方式来处理掉这些“编译过程的目标文件以及配置文件”：

```
[root@study ~]# cd /usr/src/kernels/linux-3.10.89/
[root@study linux-3.10.89]# make mrproper
```

请注意，这个动作会将你以前进行过的核心功能选择文件也删除掉， 所以几乎只有第一次执行核心编译前才进行这个动作，其余的时刻，你想要删除前一次编译过程的残留数据， 只要下达：

```
[root@study linux-3.10.89]# make clean
```

因为 make clean 仅会删除类似目标文件之类的编译过程产生的中间文件，而不会删除配置文件！很重要的！千万不要搞乱了喔！好了，既然我们是第一次进行编译，因此，请下达“make mrproper”吧！

24.2.3 开始挑选核心功能：make XXconfig

不知道你有没有发现 /boot/ 下面存在一个名为 config-xxx 的文件？那个文件其实就是核心功能列表文件！我们下面要进行的动作，其实就是作出该文件！而我们后续小节所要进行的编译动作，其实也就是通过这个文件来处理的！核心功能的挑选，最后会在 /usr/src/kernels/linux-3.10.89/ 下面产生一个名为 .config 的隐藏文件，这个文件就是 /boot/config-xxx 的文件啦！那么这个文件如何创建呢？你可以通过非常多的方法来创建这个文件！常见的方法有：[1]

- make menuconfig 最常使用的，是文字模式下面可以显示类似图形接口的方式，不需要启动 X Window 就能够挑选核心功能菜单！
- make oldconfig 通过使用已存在的 ./.config 文件内容，使用该文件内的设置值为默认值，只将新版本核心内的新功能选项列出让使用者选择，可以简化核心功能的挑选过程！对于作为升级核心源代码后的功能挑选来说，是非常好用的一个项目！
- make xconfig 通过以 Qt 为图形接口基础功能的图形化接口显示，需要具有 X window 的支持。例如 KDE 就是通过 Qt 来设计的 X Window，因此你如果在 KDE 画面中，可以使用此一项目。
- make gconfig 通过以 Gtk 为图形接口基础功能的图形化接口显示，需要具有 X window 的支持。例如 GNOME 就是通过 Gtk 来设计的 X Window，因此你如果在 GNOME 画面中，可以使用此一项目。
- make config 最旧式的功能挑选方法，每个项目都以条列式一条一条的列出让你选择，如果设置错误只能够再次选择，很不人性化啊！

大致的功能选择有上述的方法，更多的方式可以参考核心目录下的 README 文件。鸟哥个人比较偏好 make menuconfig 这个项目啦！如果你喜欢使用图形接口，然后使用鼠标去挑选所需要的功能时，也能使用 make xconfig 或 make gconfig ，不过需要有相关的图形接口支持！如果你是升级核心源代码并且需要重新编译，那么使用 make oldconfig 会比较适当！

- 通过既有的设置来处理核心项目与功能的选择

如果你跟鸟哥一样懒，那可以这样思考一下。既然我们的 CentOS 7 已经有提供它的核心设置值，我们也只是想要修改一些小细节而已，那么能不能以 CentOS 7 的核心功能为底，然后来细部微调其它的设置呢？当然可以啊！你只要这样做即可：

```
[root@study linux-3.10.89]# cp /boot/config-3.10.0-229.11.1.el7.x86_64 .config
# 上面那个版本请依据你自己的环境来填写～
```

接下来要开始调整啰！那么如何选择呢？以 make menuconfig 来说，出现的画面会有点像这样：



Tips 注意，你可能会被要求安装好多软件，请自行使用 yum 来安装喔！这里不再介绍了！另外：“不要再使用 make mrproper ”喔！因为我们已经复制了 .config 啊！使用 make mrproper 会将 .config 删除喔！



图 24.2.1、make menuconfig 核心功能挑选菜单示意图

看到上面的图示之后，你会发现画面主要分为两大部分，一个是大框框内的反白光柱，另一个则是下面的小框框， 里面有 select, exit 与 help 三个选项的内容。这几个元件的大致用法如下：

- “左右方向键”：可以移动最下面的 <Select>, <Exit>, <Help>项目；
- “上下方向键”：可以移动上面大框框部分的反白光柱，若该行有箭头 （--->） 则表示该行内部还有其他细项需要来设置的意思；
- 选定项目 ：以“上下键”选择好想要设置的项目之后，并以“左右键”选择 <Select> 之后，按下“ Enter ”就可以进入该项目去作更进一步的细部设置啰；
- 可挑选之功能：在细部项目的设置当中，如果前面有 [ ] 或 < > 符号时，该项目才可以选

择， 而选择可以使用“空白键”来选择；

- 若为 [/ <> 则表示编译进核心；若为 <M> 则表示编译成模块！ 尽量在不知道该项目为何时，且有模块可以选，那么就可以直接选择为模块啰！
- 当在细项目选择 <Exit> 后，并按下 Enter ，那么就可以离开该细部项目啰！

基本上建议只要“上下左右的方向键、空白键、Enter”这六个按键就好了！不要使用 Esc ，否则一不小心就有可能按错的！另外，关于整个核心功能的选择上面，建议你可以这样思考：

- “肯定”核心一定要的功能，直接编译进核心内；
- “可能在未来会用到”的功能，那么尽量编译成为模块；
- “不知道那个东西要干嘛的，看 help 也看不懂”的话，那么就保留默认值，或者将他编译成为模块；

总之，尽量保持核心小而美，剩下的功能就编译成为模块，尤其是“需要考虑到未来扩充性”，像鸟哥之前认为螃蟹卡就够我用的了，结果，后来竟然网站流量大增，鸟哥只好改换 3Com 的网卡。 不过，我的核心却没有相关的模块可以使用～因为.....鸟哥自己编译的核心忘记加入这个模块了。 最后，只好重新编译一次核心的模块，呵呵！真是惨痛的教训啊！

24.2.4 核心功能细项选择

由上面的图示当中，我们知道核心的可以选择的项目有很多啊！光是第一面，就有 17 个项目，每个项目内还有不同的细项！哇！真是很麻烦啊～每个项目其实都可能有 <Help> 的说明，所以，如果看到不懂的项目，务必要使用 Help 查阅查阅！ 好了，下面我们就一个一个项目来看看如何选择吧！



Tips 在下面的案例中，因为鸟哥使用的是 CentOS 7.1 的核心配置文件来进行默认的设置，所以基本上许多默认的设置都不用重新调整。下面只列出几个鸟哥认为比较重要的设置项目。其他更详细的核心功能项目，还请自行参考 help 的说明喔！

- General setup

与 Linux 最相关的程序互动、核心版本说明、是否使用发展中程序码等信息都在这里设置的。 这里的项目主要都是针对核心与程序之间的相关性来设计的，基本上，保留默认值即可！ 不要随便取消下面的任何一个项目，因为可能会造成某些程序无法被同时执行的困境喔！ 不过下面有非常多新的功能，如果你有不清楚的地方，可以按 <Help> 进入查阅，里面会有一些建议！ 你可以依据 Help 的建议来选择新功能的启动与否！

```
（vbird） Local version - append to kernel release
[*] Automatically append version information to the version string
    # 我希望我的核心版本成为 3.10.89.vbird ，那这里可以就这样设置！
    Kernel compression mode （Bzip2） ---&gt;
    # 建议选择成为 Bzip2 即可，因为压缩比较佳！
.....（其他保留默认值）.....

&lt;M&gt; Kernel .config support
[ ]   Enable access to .config through /proc/config.gz （NEW）
    # 让 .config 这个核心功能列表可以写入实际的核心文件中！所以就不需要保留 .config 文件啰！
（20） Kernel log buffer size （16 =&gt; 64KB, 17 =&gt; 128KB）
    # CentOS 7 增加了核心的登录文件容量！占用了 2 的 20 次方，大概用了 1MB 的容量！
.....（其他保留默认值）.....

[*] Initial RAM filesystem and RAM disk （initramfs/initrd） support
（）    Initramfs source file（s）
    # 这是一定要的！因为要支持开机时载入 initail RAM disk 嘛！
[  ] Optimize for size
    # 减低核心的文件大小，其实 gcc 参数使用 -Os 而不是 -O2。不过我们不是嵌入式系统，不太需要！
[ ] Configure standard kernel features （expert users） ---&gt;
[ ] Embedded system
    # 上面两个在决定是否支持嵌入式系统呢？我们这里是台式机，所以这个不用选择了！
.....（其他保留默认值）.....
```

- loadable module + block layer

要让你的核心能够支持动态的核心模块，那么下面的第一个设置就得要启动才行！至于第二个 block layer 则默认是启动的， 你也可以进入该项目的细项设置，选择其中你认为需要的功能即可！

```
[*] Enable loadable module support  ---&gt; &lt;==下面为细项
  --- Enable loadable module support
  [*]   Forced module loading
  [*]   Module unloading
  [*]     Forced module unloading  # 其实鸟哥认为这个项目可能可以选择的！免得常常无法卸载模块！
  [*]   Module versioning support
  [*]   Source checksum for all modules
  [*]   Module signature verification
  [  ]     Require modules to be validly signed
  [*]     Automatically sign all modules
        Which hash algorithm should modules be signed with? # 可以选择 SHA256 即可！
================================================================================
  -*- Enable the block layer  ---&gt;  &lt;==看吧！默认就是已经选择了！下面为细项
  -*-   Block layer SG support v4
  -*-   Block layer SG support v4 helper lib
  [*]   Block layer data integrity support
  [*]   Block layer bio throttling support
        Partition Types  ---&gt;  # 至少下面的数个项目要选择！
  [*]     Macintosh partition map support
  [*]     PC BIOS （MSDOS partition tables） support
  [*]     Windows Logical Disk Manager （Dynamic Disk） support
  [*]     SGI partition support
  [*]     EFI GUID Partition support
.....（其他保留默认值）.....

        IO Schedulers  ---&gt;  # 磁盘伫列的处理方式
  &lt;*&gt;     Deadline I/O scheduler      # 鸟哥非常建议将此项目设置为核心功能！
  &lt;*&gt;     CFQ I/O scheduler
  [*]       CFQ Group Scheduling support
          Default I/O scheduler （Deadline）  ---&gt;  # 相当建议改为Deadline
```

- CPU 的类型与功能选择

进入“Processor type and features”后，请挑选你主机的实际 CPU 形式。鸟哥这里使用的是 Intel E5 的 CPU， 而且鸟哥的主机还有启动 KVM 这个虚拟化的服务（在一部主机上面同时启动多个操作系统），因此，所以下面的选择是这样的：

```
.....（其他保留默认值）.....
  [*] Linux guest support  ---&gt;     # 提供 Linux 虚拟化功能
  [*]   Enable paravirtualization code    # 至少下面这几样一定要有选择才好！
  [*]     Paravirtualization layer for spinlocks
  [*]     Xen guest support
  [*]   KVM Guest support （including kvmclock）
  [*]   Paravirtual steal time accounting
.....（其他保留默认值）.....

      Processor family （Generic-x86-64）  ---&gt;  # 除非你是旧系统，否则就用他！
  [*] Enable Maximum number of SMP Processors and NUMA Nodes
  [*] Multi-core scheduler support
      Preemption Model （No Forced Preemption （Server））  ---&gt;  # 调整成 server 喔！原本是
.....（其他保留默认值）.....

      Timer frequency （300 HZ）  ---&gt;  # server 设置成 300 即可！
    # 这个项目则与核心针对某个事件立即回应的速度有关。Server 用途可以调整到
    # 300Hz 即可，如果是桌面电脑使用，需要调整高一点，例如 1000Hz 较佳！
.....（其他保留默认值）.....
```

- 电源管理功能

如果选择了“Power management and ACPI options”之后，就会进入系统的电源管理机制中。其实电源管理机制还需要搭配主板以及 CPU 的相关省电功能，才能够实际达到省电的效率啦！ 不论是 Server 还是 Desktop 的使用，在目前电力不足的情况下，能省电就加以省电吧！

```
.....（其他保留默认值）.....
  [*] ACPI （Advanced Configuration and Power Interface） Support  ---&gt;
    # 对嵌入式系统来说，由于可能会增加核心容量故需要考虑考虑。至于 desktop/server 当然就选择啊
    # 至于内容细项大致保持默认值即可
    CPU Frequency scaling  ---&gt;
    # 决定 CPU 频率的一个重要项目，基本上的项目是 ondemand 与 performance 两者！
    &lt;M&gt;   CPU frequency translation statistics
    [*]     CPU frequency translation statistics details
          Default CPUFreq governor （ondemand）  ---&gt;  # 现在大家都建议用这个！
    -*-   'performance' governor
    &lt;*&gt;   'powersave' governor
    &lt;*&gt;   'userspace' governor for userspace frequency scaling
    -*-   'ondemand' cpufreq policy governor
    &lt;*&gt;   'conservative' cpufreq governor
          x86 CPU frequency scaling drivers  ---&gt;
          # 这个子项目内全部都是省电机制，能编成模块的全部选择！要加入核心的都加入就对了！
```

- 一些总线（bus）的选项

这个“Bus options （PCI etc.）”项目则与总线有关啦！分为最常见的 PCI 与 PCI-express 的支持，还有笔记本电脑常见的 PCMCIA 插卡啊！要记住的是，那个 PCI-E 的接口务必要选取！不然你的新显卡可能会捉不到！

```
[*] PCI support
[*]   Support mmconfig PCI config space access
[*]   PCI Express support
&lt;*&gt;     PCI Express Hotplug driver
.....（其他在 PCI Express 下面的项目大多保留默认值）.....
-*- Message Signaled Interrupts （MSI and MSI-X）
&lt;*&gt; PCI Stub driver   # 如果要玩虚拟化，这个部份建议编进核心！
.....（其他保留默认值）.....
```

- 编译后可执行文件的格式

选择“Executable file formats / Emulations”会见到如下选项。 下面的选项必须要勾选才行喔！因为是给 Linux 核心运行可执行文件之用的数据。通常是与编译行为有关啦！

```
-*- Kernel support for ELF binaries
[*] Write ELF core dumps with partial segments
&lt;*&gt; Kernel support for scripts starting with #!
&lt;M&gt; Kernel support for MISC binaries
[*] IA32 Emulation
&lt;M&gt;   IA32 a.out support
[*]   x32 ABI for 64-bit mode
# 因为我们的 CentOS 已经是纯 64 位的环境！所以个人建议这里还是要选择仿真 32 位的功能！
# 不然若有些比较旧的软件，恐怕会无法被你的系统所执行喔！
```

- 核心的网络功能

这个“Networking support”项目是相当重要的选项，因为他还包含了防火墙相关的项目！就是未来在服务器篇会谈到的防火墙 iptables 这个数据啊！所以，千万注意了！在这个设置项目当中，很多东西其实我们在基础篇还没有讲到， 因为大部分的参数都与网络、防火墙有关！由于防火墙是在启动网络之后再设置即可，所以绝大部分的内容都可以被编译成为模块，而且也建议你编成模块！有用到再载入到核心即可啊！

```
--- Networking support
      Networking options  ---&gt;
      # 就是这个光啊！里面的数据全部都是重要的防火墙项目！尽量编成模块啰！
      # 至于不晓得功能的部分，就尽量保留默认值即可！
      # 下面的数据中，鸟哥只有列出原本没有选择，后来建议选择的部份
      [*] Network packet filtering framework （Netfilter）  ---&gt;
      # 这个就是我们一直讲的防火墙部分！里面细项几乎全选择成为模块！
          --- Network packet filtering framework （Netfilter）
                Core Netfilter Configuration  ---&gt;
                &lt;M&gt; Transparent proxying support
================================================================================
      [*] QoS and/or fair queueing  ---&gt; &lt;==内容同样全为模块！
          Network testing  ---&gt; &lt;==保留成模块默认值
================================================================================
# 下面的则是一些特殊的网络设备，例如红外线啊、蓝牙啊！
# 如果不清楚的话，就使用模块吧！除非你真的知道不要该项目！
&lt;M&gt;   Bluetooth subsystem support  ---&gt;
      # 这个是蓝牙支持，同样的，里面除了必选之外，其他通通挑选成为模块！
[*]   Wireless  ---&gt;
      # 这个则是无线网络设备，里面保留默认值，但可编成模块的就选模块
&lt;M&gt;   WiMAX Wireless Broadband support  ---&gt;
      # 新一代的无线网络，也请勾选成为模块！
&lt;M&gt;  NFC subsystem support  ---&gt;
      # 跟卡片比较有关的芯片支持，建议编译成模块，内部数据也是编译成模块为佳！
```

- 各项设备的驱动程序

进入“Device Drivers”这个是所有硬件设备的驱动程序库！哇！光是看到里面这么多内容，鸟哥头都昏了～ 不过，为了你自己的主机好，建议你还是得要一个项目一个项目的去挑选挑选才行～ 这里面的数据就与你主机的硬件有绝对的关系了！

在这里面真的很重要，因为很多数据都与你的硬件有关。核心推出时的默认值是比较符合一般状态的， 所以很多数据其实保留默认值就可以编的很不错了！不过，也因为较符合一般状态， 所以核心额外的编译进来很多跟你的主机系统不符合的数据，例如网卡设备～ 你可以针对你的主板与相关硬件来进行编译。不过，还是要记得有“未来扩充性”的考虑！ 之前鸟哥不是谈过吗，我的网卡由螃蟹卡换成 3Com 时，核心捉不到～ 因为...鸟哥并没有将 3Com 的网卡编译成为模块啊！ @_@

```
# 大部分都保留默认值，鸟哥只是就比较重要的部份拿出来做说明而已！
    &lt;M&gt; Serial ATA and Parallel ATA drivers  ---&gt;  # 就是 SATA/IDE 磁盘！大多数选择
    [*] Multiple devices driver support （RAID and LVM）  ---&gt; # 就是 LVM 与 RAID ！要选
    -*- Network device support  ---&gt; # 网络方面的设备，网卡与相关媒体啦！
        -*-   Network core driver support
        &lt;M&gt;     Bonding driver support             # 与网卡整合有关的项目！要选！
        &lt;M&gt;     Ethernet team driver support  ---&gt; # 与 bonding 差不多的功能！要选！
        &lt;M&gt;     Virtio network driver              # 虚拟化的网卡驱动程序！要选！
        -*-   Ethernet driver support  ---&gt;       # 以太网卡！里面的一堆 10G 卡要选！
              &lt;M&gt;     Chelsio 10Gb Ethernet support
              &lt;M&gt;     Intel（R） PRO/10GbE support
        &lt;M&gt;   PPP （point-to-point protocol） support# 与拨接有关的协定！
              USB Network Adapters  ---&gt;          # 当然全部编译为模块！
        [*]   Wireless LAN  ---&gt;                  # 无线网卡也相当重要！里面全部变成模块！
================================================================================
    [ ] GPIO Support  ---&gt;           # 若有需要使用类似树莓派、香蕉派才需要这东西！
    &lt;M&gt; Multimedia support  ---&gt;      # 多媒体设备，如影像撷取、广播声卡等等
        Graphics support  ---&gt;       # 显卡！如果是作为桌上型使用，这里就重要了！
    &lt;M&gt; Sound card support  ---&gt;      # 声卡，同样的，桌面电脑使用时，比较重要！
    [*] USB support  ---&gt;            # 就是 USB！下面几个内部的细项要注意是勾选的！
        &lt;*&gt;     xHCI HCD （USB 3.0） support
        &lt;*&gt;     EHCI HCD （USB 2.0） support
        &lt;*&gt;     OHCI HCD support
        &lt;*&gt;     UHCI HCD （most Intel and VIA） support
    &lt;M&gt; InfiniBand support  ---&gt;      # 较高阶的网络设备，速度通常达到 40Gb 以上！
    &lt;M&gt; VFIO Non-Privileged userspace driver framework  ---&gt; # 作为 VGA passthrou
        [*]     VFIO PCI support for VGA devices
    [*] Virtualization drivers  ---&gt; # 虚拟化的驱动程序！
        Virtio drivers  ---&gt;         # 在虚拟机里面很重要的驱动程序项目！
    [*] IOMMU Hardware Support  ---&gt; # 同样的与虚拟化相关性较高！
```

至于“ Firmware Drivers ”的项目，请视你的需求来选择～基本上就保留设置值即可！所以鸟哥这里就不显示啰！

- 文件系统的支持

文件系统的支持也是很重要的一项核心功能！因为如果不支持某个文件系统，那么我们的 Linux kernel 就无法认识，当然也就无法使用啦！例如 Quota, NTFS 等等特殊的 filesystem 。 这部份也是有够麻烦～因为涉及核心是否能够支持某些文件系统，以及某些操作系统支持的 partition table 项目。在进行选择时，也务必要特别的小心在意喔！ 尤其是我们常常用到的网络操作系统 （NFS/Samba 等等），以及基础篇谈到的 Quota 等， 你都得要勾选啊！否则是无法被支持的。如果你有兴趣，也可以将 NTFS 的文件系统设置为可读写看看啰！

```
# 下面仅有列出比较重要及与默认值不同的项目而已喔！所以项目少很多！
    &lt;M&gt; Second extended fs support           # 默认已经不支持 ext2/ext3，这里我们将他加回
    &lt;M&gt; Ext3 journalling file system support
    [*]   Default to 'data=ordered' in ext3 （NEW）
    [*]   Ext3 extended attributes （NEW）
    [*]     Ext3 POSIX Access Control Lists
    &lt;M&gt; The Extended 4 （ext4） filesystem     # 一定要有的支持
    &lt;M&gt; Reiserfs support
    &lt;M&gt; XFS filesystem support               # 一定要有的支持！
    [*]   XFS Quota support
    [*]   XFS POSIX ACL support
    [*]   XFS Realtime subvolume support     # 增加这一项好了！
    &lt;M&gt; Btrfs filesystem support             # 最好有支持！
    [*] Quota support
    &lt;*&gt; Quota format vfsv0 and vfsv1 support
    &lt;*&gt; Kernel automounter version 4 support （also supports v3）
    &lt;M&gt; FUSE （Filesystem in Userspace） support
        DOS/FAT/NT Filesystems  ---&gt;
        &lt;M&gt; MSDOS fs support
        &lt;M&gt; VFAT （Windows-95） fs support
        （950） Default codepage for FAT       # 要改成这样喔！中文支持！
        （utf8） Default iocharset for FAT     # 要改成这样喔！中文支持！
        &lt;M&gt; NTFS file system support         # 建议加上 NTFS 喔！
        [*]   NTFS write support             # 让他可读写好了！
        Pseudo filesystems  ---&gt;             # 类似 /proc ，保留默认值
    -*- Miscellaneous filesystems  ---&gt;      # 其他文件系统的支持，保留默认值
    [*] Network File Systems  ---&gt;           # 网络文件系统！很重要！也要挑挑！
        &lt;M&gt;   NFS client support
        &lt;M&gt;   NFS server support
        [*]     NFS server support for NFS version 4
        &lt;M&gt;   CIFS support （advanced network filesystem, SMBFS successor）
        [*]       Extended statistics
        [*]   Provide CIFS client caching support
    -*- Native language support  ---&gt;        # 选择默认的语系
        （utf8） Default NLS Option
        &lt;M&gt;   Traditional Chinese charset （Big5）
```

- 核心骇客、信息安全、密码应用

再接下来有个“Kernel hacking”的项目，那是与核心开发者比较有关的部分，这部分建议保留默认值即可， 应该不需要去修改他！除非你想要进行核心方面的研究喔。然后下面有个“ Security Options ”，那是属于信息安全方面的设置， 包括 SELinux 这个细部权限强化模块也在这里编入核心的！这个部份只要记得 SELinux 作为默认值，且务必要将 NSA SELinux 编进核心即可， 其他的细部请保留默认值。

另外还有“ Cryptographic API ”这个密码应用程序接口工具选项，以前的默认加密机制为 MD5，近年来则改用了 SHA 这种机制。 不过，反正默认已经将所有的加密机制编译进来了，所以也是可以保留默认值啦！都不需要额外修改就是了！

- 虚拟化与函数库

虚拟化是近年来非常热门的一个议题，因为计算机的能力太强，所以时常闲置在那边， 此时，我们可以通过虚拟化技术在一部主机上面同时启动多个操作系统来运行，这就是所谓的虚拟化。 Linux 核心已经主动的纳入虚拟化功能喔！而 Linux 认可的虚拟化使用的机制为 KVM （Kernel base Virtual Machine）。 至于常用的核心函数库也可以全部编为模块啰！

```
[*] Virtualization  ---&gt;
    --- Virtualization
    &lt;M&gt;   Kernel-based Virtual Machine （KVM） support
    &lt;M&gt;     KVM for Intel processors support
    &lt;M&gt;     KVM for AMD processors support
    [*]     Audit KVM MMU
    [*]     KVM legacy PCI device assignment support   # 虽然已经有 VFIO，不过建议还是选起来！
    &lt;M&gt;   Host kernel accelerator for virtio net
================================================================================
Library routines  ---&gt;
    # 这部份全部保留默认值即可！
```

现在请回到如图24.2.1 的画面中，在下方设置处移动到“Save”的选项，点选该项目， 在出现的窗口中确认文件名为 .config 之后，直接按下“OK”按钮，这样就将刚刚处理完毕的选项给记录下来了。 接下来可以选择离开菜单画面，准备让我们来进行编译的行为啰。

要请你注意的是，上面的数据主要是适用在鸟哥的个人机器上面的， 目前鸟哥比较习惯使用原本 distributions 提供的默认核心，因为他们也会主动的进行更新， 所以鸟哥就懒的自己重编核心了～ ^_^

此外，因为鸟哥重视的地方在于“网络服务器与虚拟化服务器”上面，所以里头的设置少掉了相当多的个人桌上型 Linux 的硬件编译！所以，如果你想要编译出一个适合你的机器的核心，那么可能还有相当多的地方需要来修正的！不论如何，请随时以 Help 那个选项来看一看内容吧！反正 Kernel 重编的概率不大！花多一点时间重新编译一次！ 然后将该编译完成的参数文件储存下来，未来就可以直接将该文件叫出来读入了！ 所以花多一点时间安装一次就好！那也是相当值得的！

24.3 核心的编译与安装

将最复杂的核心功能选择完毕后，接下来就是进行这些核心、核心模块的编译了！而编译完成后，当然就是需要使用噜～ 那如何使用新核心呢？就得要考虑 grub 这个玩意儿啦！下面我们就来处理处理：

24.3.1 编译核心与核心模块

核心与核心模块需要先编译起来，而编译的过程其实非常简单，你可以先使用“ make help ”去查阅一下所有可用编译参数， 就会知道有下面这些基本功能：

```
[root@study linux-3.10.89]# make vmlinux  &lt;==未经压缩的核心
[root@study linux-3.10.89]# make modules  &lt;==仅核心模块
[root@study linux-3.10.89]# make bzImage  &lt;==经压缩过的核心（默认）
[root@study linux-3.10.89]# make all      &lt;==进行上述的三个动作
```

我们常见的在 /boot/ 下面的核心文件，都是经过压缩过的核心文件，因此，上述的动作中比较常用的是 modules 与 bzImage 这两个，其中 bzImage 第三个字母是英文大写的 I 喔！ bzImage 可以制作出压缩过后的核心， 也就是一般我们拿来进行系统开机的信息啰！所以，基本上我们会进行的动作是：

```
[root@study linux-3.10.89]# make -j 4 clean    &lt;==先清除暂存盘
[root@study linux-3.10.89]# make -j 4 bzImage  &lt;==先编译核心
[root@study linux-3.10.89]# make -j 4 modules  &lt;==再编译模块
[root@study linux-3.10.89]# make -j 4 clean bzImage modules  &lt;==连续动作！
```

上述的动作会花费非常长的时间，编译的动作依据你选择的项目以及你主机硬件的性能而不同。此外，为啥要加上 -j 4 呢？ 因为鸟哥的系统上面有四个 CPU 核心，这几个核心可以同时进行编译的行为，这样在编译时速度会比较快！如果你的 CPU 核心数 （包括超线程） 有多个， 那这个地方请加上你的可用 CPU 数量吧！

最后制作出来的数据是被放置在 /usr/src/kernels/linux-3.10.89/ 这个目录下，还没有被放到系统的相关路径中喔！在上面的编译过程当中，如果有发生任何错误的话， 很可能是由于核心项目的挑选选择的不好，可能你需要重新以 make menuconfig 再次的检查一下你的相关设置喔！ 如果还是无法成功的话，那么或许将原本的核心数据内的 .config 文件，复制到你的核心原始文件目录下， 然后据以修改，应该就可以顺利的编译出你的核心了。最后注意到，下达了 make bzImage 后，最终的结果应该会像这样：

```
Setup is 16752 Bytes （padded to 16896 Bytes）.
System is 4404 kB
CRC 30310acf
Kernel: arch/x86/boot/bzImage is ready  （#1）

[root@study linux-3.10.89]# ll arch/x86/boot/bzImage
-rw-r--r--. 1 root root 4526464 Oct 20 09:09 arch/x86/boot/bzImage
```

可以发现你的核心已经编译好而且放置在 /usr/src/kernels/linux-3.10.89/arch/x86/boot/bzImage 里面啰～那个就是我们的核心文件！最重要就是他啦！我们等一下就会安装到这个文件哩！ 然后就是编译模块的部分啰～ make modules 进行完毕后，就等着安装啦！ ^_^

24.3.2 实际安装模块

安装模块前有个地方得要特别强调喔！我们知道模块是放置到 /lib/modules/$（uname -r） 目录下的，那如果同一个版本的模块被反复编译后来安装时，会不会产生冲突呢？举例来说，鸟哥这个 3.10.89 的版本第一次编译完成且安装妥当后，发现有个小细节想要重新处理，因此又重新编译过一次，那两个版本一模一样时， 模块放置的目录会一样，此时就会产生冲突了！如何是好？有两个解决方法啦：

- 先将旧的模块目录更名，然后才安装核心模块到目标目录去；
- 在 make menuconfig 时，那个 General setup 内的 Local version 修改成新的名称。

鸟哥建议使用第二个方式，因为如此一来，你的模块放置的目录名称就不会相同，这样也就能略过上述的目录同名问题啰！ 好，那么如何安装模块到正确的目标目录呢？很简单，同样使用 make 的功能即可：

```
[root@study linux-3.10.89]# make modules_install
[root@study linux-3.10.89]# ll /lib/modules/
drwxr-xr-x. 7 root root 4096 Sep  9 01:14 3.10.0-229.11.1.el7.x86_64
drwxr-xr-x. 7 root root 4096 May  4 17:56 3.10.0-229.el7.x86_64
drwxr-xr-x. 3 root root 4096 Oct 20 14:29 3.10.89vbird  # 这就是刚刚装好的核心模块！
```

看到否，最终会在 /lib/modules 下面创建起你这个核心的相关模块喔！不错吧！模块这样就已经处理妥当啰～ 接下来，就是准备要进行核心的安装了！哈哈！又跟 grub2 有关啰～

24.3.3 开始安装新核心与多重核心菜单 （grub）

现在我们知道核心文件放置在 /usr/src/kernels/linux-3.10.89/arch/x86/boot/bzImage ，但是其实系统核心理论上都是摆在 /boot 下面，且为 vmlinuz 开头的文件名。 此外，我们也晓得一部主机是可以做成多重开机系统的！这样说，应该知道鸟哥想要干嘛了吧？ 对啦！我们将同时保留旧版的核心，并且新增新版的核心在我们的主机上面。

此外，与 grub1 不一样， grub2 建议我们不要直接修改配置文件，而是通过让系统自动侦测来处理 grub.cfg 这个配置文件的内容。 所以，在处理核心文件时，可能就得要知道核心文件的命名规则比较好耶！

- 移动核心到 /boot 且保留旧核心文件

保留旧核心有什么好处呢？最大的好处是可以确保系统能够顺利开机啦！因为核心虽然被编译成功了， 但是并不保证我们刚刚挑选的核心项目完全适合于目前这部主机系统， 可能有某些地方我们忘记选择了，这将导致新核心无法顺利驱动整个主机系统，更差的情况是， 你的

主机无法成功开机成功！此时，如果我们保留旧的核心，呵呵！若新核心测试不通过，就用旧核心来启动啊！嘿嘿！ 保证比较不会有问题嘛！另外，核心文件通常以 vmlinuz 为开头，接上核心版本为依据的文件名格式，因此可以这样做看看：

```
[root@study linux-3.10.89]# cp arch/x86/boot/bzImage /boot/vmlinuz-3.10.89vbird  &lt;==实
[root@study linux-3.10.89]# cp .config /boot/config-3.10.89vbird   &lt;==建议配置文件也复制备
[root@study linux-3.10.89]# chmod a+x /boot/vmlinuz-3.10.89vbird
[root@study linux-3.10.89]# cp System.map /boot/System.map-3.10.89vbird
[root@study linux-3.10.89]# gzip -c Module.symvers &gt; /boot/symvers-3.10.89vbird.gz
[root@study linux-3.10.89]# restorecon -Rv /boot
```

- 创建相对应的 Initial Ram Disk （initrd）

还记得第十九章谈过的 initramfs 这个玩意儿吧！ 由于鸟哥的系统使用 SATA 磁盘，加上刚刚 SATA 磁盘支持的功能并没有直接编译到核心去，所以当然要使用 initramfs 来载入才行！ 使用如下的方法来创建 initramfs 吧！记得搭配正确的核心版本喔！

```
[root@study ~]# dracut -v /boot/initramfs-3.10.89vbird.img 3.10.89vbird
```

- 编辑开机菜单 （grub）

前面的文件大致上都摆放妥当之后，同时得要依据你的核心版本来处理文件名喔！接下来就直接使用 grub2-mkconfig 来处理你的 grub2 开机菜单设置即可！让我们来处理处理先！

```
[root@study ~]# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.10.89vbird     # 应该要最早出现！
Found initrd image: /boot/initramfs-3.10.89vbird.img
.....（下面省略）.....
```

因为默认较新版本的核心会放在最前面成为默认的开机菜单项目，所以你得要确认上述的结果中，第一个被发现的核心为你刚刚编译好的核心文件才对喔！ 否则等一下开机可能就会出现使用旧核心开机的问题。现在让我们重新开机来测试看看啰！

- 重新以新核心开机、测试、修改

如果上述的动作都成功后，接下来就是重新开机并选择新核心来启动系统啦！如果系统顺利启动之后，你使用 uname -a 会出现类似下面的数据：

```
[root@study ~]# uname -a
Linux study.centos.vbird 3.10.89vbird #1 SMP Tue Oct 20 09:09:11 CST 2015 x86_64
x86_64 x86_64 GNU/Linux
```

包括核心版本与支持的硬件平台都是 OK 的！嘿嘿！那你所编译的核心就是差不多成功的啦！ 如果运行一阵子后，你的系统还是稳定的情况下，那就能够将 default 值使用这个新的核心来作为默认开机啰！ 这就是核心编译！那你也可以自己处理嵌入式系统的核心编译啰！ ^_^

24.4 额外（单一）核心模块编译

我们现在知道核心所支持的功能当中，有直接编译到核心内部的，也有使用外挂模块的，外挂模块可以简单的想成就是驱动程序 啦！那么也知道这些核心模块依据不同的版本，被分别放置到 /lib/modules/$（uname -r）/kernel/ 目录中，各个硬件的驱动程序则是放置到 /lib/modules/$（uname -r）/kernel/drivers/ 当中！换个角度再来思考一下，如果刚刚我自己编译的数据中，有些驱动程序忘记编译成为模块了，那是否需要重新进行上述的所有动作？又如果我想要使用硬件厂商释出的新驱动程序，那该如何是好？

24.4.1 编译前注意事项

由于我们的核心原本就有提供很多的核心工具给硬件开发商来使用， 而硬件开发商也需要针对核心所提供的功能来设计他们的驱动程序模块，因此， 我们如果想要自行使用硬件开发商所提供的模块来进行编译时，就需要使用到核心所提供的原始文件当中， 所谓的头文件案（header include file） 来取得驱动模块所需要的一些函数库或标头的定义啦！ 也因此我们常常会发现到，如果想要自行编译核心模块时，就得要拥有核心源代码嘛！

那核心源代码我们知道他是可能放置在 /usr/src/ 下面，早期的核心源代码被要求一定要放置到 /usr/src/linux/ 目录下，不过，如果你有多个核心在一个 Linux 系统当中，而且使用的源代码并不相同时， 呵呵～问题可就大了！所以，在 2.6 版以后，核心使用比较有趣的方法来设计他的源代码放置目录， 那就是以 /lib/modules/$（uname -r）/build 及 /lib/modules/$（uname -r）/source 这两个链接文件来指向正确的核心源代码放置目录。如果以我们刚刚由 kernel 3.10.89vbird 创建的核心模块来说， 那么他的核心模块目录下面有什么咚咚？

```
[root@study ~]# ll -h /lib/modules/3.10.89vbird/
lrwxrwxrwx.  1 root root   30 Oct 20 14:27 build -&gt; /usr/src/kernels/linux-3.10.89
drwxr-xr-x. 11 root root 4.0K Oct 20 14:29 kernel
-rw-r--r--.  1 root root 668K Oct 20 14:29 modules.alias
-rw-r--r--.  1 root root 649K Oct 20 14:29 modules.alias.bin
-rw-r--r--.  1 root root 5.8K Oct 20 14:27 modules.builtin
-rw-r--r--.  1 root root 7.5K Oct 20 14:29 modules.builtin.bin
-rw-r--r--.  1 root root 208K Oct 20 14:29 modules.dep
-rw-r--r--.  1 root root 301K Oct 20 14:29 modules.dep.bin
-rw-r--r--.  1 root root  316 Oct 20 14:29 modules.devname
-rw-r--r--.  1 root root  81K Oct 20 14:27 modules.order
-rw-r--r--.  1 root root  131 Oct 20 14:29 modules.softdep
-rw-r--r--.  1 root root 269K Oct 20 14:29 modules.symbols
-rw-r--r--.  1 root root 339K Oct 20 14:29 modules.symbols.bin
lrwxrwxrwx.  1 root root   30 Oct 20 14:27 source -&gt; /usr/src/kernels/linux-3.10.89
```

比较有趣的除了那两个链接文件之外，还有那个 modules.dep 文件也挺有趣的， 那个文件是记录了核心模块的相依属性的地方，依据该文件，我们可以简单的使用 modprobe 这个指令来载入模块呢！至于核心源代码提供的头文件，在上面的案例当中， 则是放置到 /usr/src/kernels/linux-3.10.89/include/ 目录中，当然就是借由 build/source 这两个链接文件来取得目录所在的啦！^_^

由于核心模块的编译其实与核心原本的源代码有点关系的，因此如果你需要重新编译模块时， 那除了 make, gcc 等主要的编译软件工具外，你还需要的就是 kernel-devel 这个软件！ 记得一定要安装喔！而如果你想要在默认的核心下面新增模块的话，那么就得要找到 kernel 的 SRPM 文件了！ 将该文件给他安装，并且取得 source code 后，才能够顺利的编译喔！

24.4.2 单一模块编译

想像两个情况：

- 如果我的默认核心忘记加入某个功能，而且该功能可以编译成为模块，不过， 默认核心却也没有将该项功能编译成为模块，害我不能使用时，该如何是好？
- 如果 Linux 核心源代码并没有某个硬件的驱动程序 （module） ，但是开发该硬件的厂商有提供给 Linux 使用的驱动程序源代码，那么我又该如何将该项功能编进核心模块呢？

很有趣对吧！不过，在这样的情况下其实没有什么好说的，反正就是“去取得源代码后，重新编译成为系统可以载入的模块”啊！很简单，对吧！^_^！ 但是，上面那两种情况的模块编译行为是不太一样的，不过，都是需要 make, gcc 以及核心所提供的 include 头文件与函数库等等。

- 硬件开发商提供的额外模块

很多时候，可能由于核心默认的核心驱动模块所提供的功能你不满意，或者是硬件开发商所提供的核心模块具有更强大的功能， 又或者该硬件是新的，所以默认的核心并没有该硬件的驱动模块时，那你只好自行由硬件开发商处取得驱动模块，然后自行编译啰！

如果你的硬件开发商有提供驱动程序的话，那么真的很好解决，直接下载该源代码，重新编译， 将他放置到核心模块该放置的地方后就能够使用了！举个例子来说，鸟哥在 2014 年底帮厂商制作一个服务器的环境时， 发现对方喜欢使用的磁盘阵列卡 （RAID） 当时并没有被 Linux 核心所支持，所以就得要帮厂商针对该磁盘阵列卡来编译成为模块啰！ 处理的方式，当然就是使用磁盘阵列卡官网提供的驱动程序来编译啰！

- Highpoint 的 RocketRAID RR640L 驱动程序： http://www.highpoint-tech.com/USA_new/series_rr600-download.htm

虽然你可以选择“RHEL/CentOS 7 x86_64”这个已编译的版本来处理，不过因为我们的核心已经做成自订的版本， 变成 3.10.89vbird 这样，忘记加上 x86_64 的版本名，会导致该版本的自动安装脚本失败！所以，算了！我们自己来重新编译吧！ 因此，请下载“Open Source Driver”的版本喔！同时，鸟哥假设你将下载的文件放置到 /root/raidcard 目录内喔！

```
# 1\. 将文件解压缩并且开始编译：
[root@study ~]# cd /root/raidcard
[root@study raidcard]# ll
-rw-r--r--. 1 root root 501477 Apr 23 07:42 RR64xl_Linux_Src_v1.3.9_15_03_07.tar.gz
[root@study raidcard]# tar -zxvf RR64xl_Linux_Src_v1.3.9_15_03_07.tar.gz
[root@study raidcard]# cd rr64xl-linux-src-v1.3.9/product/rr64xl/linux/
[root@study linux]# ll
-rw-r--r--. 1 dmtsai dmtsai 1043 Mar  7  2015 config.c
-rwxr-xr-x. 1 dmtsai dmtsai  395 Dec 27  2013 Makefile      # 要有这家伙存在才行！
[root@study linux]# make
make[1]: Entering directory `/usr/src/kernels/linux-3.10.89'
  CC [M]  /root/raidcard/rr64xl-linux-src-v1.3.9/product/rr64xl/linux/.build/os_linux.o
  CC [M]  /root/raidcard/rr64xl-linux-src-v1.3.9/product/rr64xl/linux/.build/osm_linux.o
.....（中间省略）.....
  LD [M]  /root/raidcard/rr64xl-linux-src-v1.3.9/product/rr64xl/linux/.build/rr640l.ko
make[1]: Leaving directory `/usr/src/kernels/linux-3.10.89'

[root@study linux]# ll
-rw-r--r--. 1 dmtsai dmtsai    1043 Mar  7  2015 config.c
-rwxr-xr-x. 1 dmtsai dmtsai     395 Dec 27  2013 Makefile
-rw-r--r--. 1 root   root   1399896 Oct 21 00:59 rr640l.ko  # 就是产生这家伙！

# 2\. 将模块放置到正确的位置去！
[root@study linux]# cp rr640l.ko /lib/modules/3.10.89vbird/kernel/drivers/scsi/
[root@study linux]# depmod -a   # 产生模块相依性文件！
[root@study linux]# grep rr640 /lib/modules/3.10.89vbird/modules.dep
kernel/drivers/scsi/rr640l.ko:  # 确定模块有在相依性的配置文件中！

[root@study linux]# modprobe rr640l
modprobe: ERROR: could not insert 'rr640l': No such device
# 要测试载入一下才行，不过，我们实际上虚拟机没有这张 RAID card，所以出现错误是正常的啦！

# 3\. 若开机过程中就得要载入此模块，则需要将模块放入 initramfs 才行喔！
[root@study linux]# dracut --force -v --add-drivers rr640l \
&gt; /boot/initramfs-3.10.89vbird.img 3.10.89vbird
[root@study linux]# lsinitrd /boot/initramfs-3.10.89vbird.img &#124; grep rr640
```

通过这样的动作，我们就可以轻易的将模块编译起来，并且还可以将他直接放置到核心模块目录中， 同时以 depmod 将模块创建相关性，未来就能够利用 modprobe 来直接取用啦！ 但是需要提醒你的是，当自行编译模块时， 若你的核心有更新 （例如利用自动更新机制进行线上更新） 时，则你必须要重新编译该模块一次， 重复上面的步骤才行！因为这个模块仅针对目前的核心来编译的啊！对吧！

- 利用旧有的核心源代码进行编译

如果你后来发现忘记加入某个模块功能了，那该如何是好？其实如果仅是重新编译模块的话， 那么整个过程就会变的非常简单！我们先到目前的核心源代码所在目录下达 make menuconfig ， 然后将 NTFS 的选项设置成为模块，之后直接下达：

> make fs/ntfs/

那么 ntfs 的模块 （ntfs.ko） 就会自动的被编译出来了！ 然后将该模块复制到 /lib/modules/3.10.89vbird/kernel/fs/ntsf/ 目录下， 再执行 depmod -a ，呵呵～就可以在原来的核心下面新增某个想要加入的模块功能啰～ ^_^

24.4.3 核心模块管理

核心与核心模块是分不开的，至于驱动程序模块在编译的时候，更与核心的源代码功能分不开～ 因此，你必须要先了解到：核心、核心模块、驱动程序模块、核心源代码与头文件案的相关性， 然后才有办法了解到为何编译驱动程序的时候老是需要找到核心的源代码才能够顺利编译！ 然后也才会知道，为何当核心更新之后，自己之前所编译的核心模块会失效～

此外，与核心模块有相关的，还有那个很常被使用的 modprobe 指令， 以及开机的时候会读取到的模块定义数据文件 /etc/modprobe.conf ， 这些数据你也必须要了解才行～相关的指令说明我们已经在第十九章内谈过了， 你应该要自行前往了解喔！ ^_^

24.5 以最新核心版本编译 CentOS 7.x 的核心

如果你跟鸟哥一样，曾经为了某些缘故需要最新的 4.x.y 的核心版本来实作某些特定的功能时，那该如何是好？没办法，只好使用最新的核心版本来编译啊！ 你可以依照上面的程序来一个一个处理，没有问题～不过，你也可以根据 ELRepo 网站提供的 SRPM 来重新编译打包喔！ 当然你可以直接使用 ELRepo 提供的 CentOS 7.x 专属的核心来直接安装。

下面我们使用 ELRepo 网站提供的 SRPM 文件来实作核心编译。而要这么重新编译的原因是，鸟哥需要将 VFIO 的 VGA 直接支持的核心功能打开！ 因此整个程序会变成类似这样：

1. 先从 ELRepo 网站下载不含源代码的 SRPM 文件，并且安装该文件
2. 从 www.kernel.org 网站下载满足 ELRepo 网站所需要的核心版本来处理
3. 修改核心功能
4. 通过 SRPM 的 rpmbuild 重新编译打包核心

就让我们来测试一下啰！（注意，鸟哥使用的是 2015/10/20 当下最新的 4.2.3 这一版的核心。由于核心版本的升级太快，因此在你实作的时间， 可能已经有更新的核心版本了。此时你应该要前往 ELRepo 查阅最新的 SRPM 之后，再决定你想使用的版本喔！）

```
1\. 先下载 ELRepo 上面的 SRPM 文件！同时安装它：
[root@study ~]# wget http://elrepo.org/linux/kernel/el7/SRPMS/kernel-ml-4.2.3-1.el7.elrep
[root@study ~]# rpm -ivh kernel-ml-4.2.3-1.el7.elrepo.nosrc.rpm

2\. 根据上述的文件，下载正确的核心源代码：
[root@study ~]# cd rpmbuild/SOURCES
[root@study SOURCES]# wget https://cdn.kernel.org/pub/linux/kernel/v4.x/linux-4.2.3.tar.x
[root@study SOURCES]# ll -tr
.....（前面省略）.....
-rw-r--r--. 1 root root 85523884 Oct  3 19:58 linux-4.2.3.tar.xz  # 核心源代码
-rw-rw-r--. 1 root root      294 Oct  3 22:04 cpupower.service
-rw-rw-r--. 1 root root      150 Oct  3 22:04 cpupower.config
-rw-rw-r--. 1 root root   162752 Oct  3 22:04 config-4.2.3-x86_64 # 主要的核心功能

3\. 修改核心功能设置：
[root@study SOURCES]# vim config-4.2.3-x86_64
# 大约在 5623 行找到下面这一行，并在下面新增一行设置值！
# CONFIG_VFIO_PCI_VGA is not set
CONFIG_VFIO_PCI_VGA=y

[root@study SOURCES]# cd ../SPECS
[root@study SPECS]# vim kernel-ml-4.2.spec
# 大概在 145 左右找到下面这一行：
Source0: ftp://ftp.kernel.org/pub/linux/kernel/v4.x/linux-%{LKAver}.tar.xz
# 将它改成如下的模样：
Source0: linux-%{LKAver}.tar.xz

4\. 开始编译并打包：
[root@study SPECS]# rpmbuild -bb kernel-ml-4.2.spec
# 接下来会有很长的一段时间在进行编译行为，鸟哥的机器曾经跑过两个小时左右才编译完！
# 所以，请耐心等候啊！
Wrote: /root/rpmbuild/RPMS/x86_64/kernel-ml-4.2.3-1.el7.centos.x86_64.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/kernel-ml-devel-4.2.3-1.el7.centos.x86_64.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/kernel-ml-headers-4.2.3-1.el7.centos.x86_64.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/perf-4.2.3-1.el7.centos.x86_64.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/python-perf-4.2.3-1.el7.centos.x86_64.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/kernel-ml-tools-4.2.3-1.el7.centos.x86_64.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/kernel-ml-tools-libs-4.2.3-1.el7.centos.x86_64.rpm
Wrote: /root/rpmbuild/RPMS/x86_64/kernel-ml-tools-libs-devel-4.2.3-1.el7.centos.x86_64.rp
```

如上表最后的状态，你会发现竟然已经有 kernel-ml 的软件包产生了！接下来你也不需要像手动安装核心一样，得要一个一个项目移动到正确的位置去， 只要使用 yum install 新的核心版本，就会有 4.2.3 版的核心在你的 CentOS 7.x 当中了耶！相当神奇！

```
[root@study ~]# yum install /root/rpmbuild/RPMS/x86_64/kernel-ml-4.2.3-1.el7.centos.x86_6
[root@study ~]# reboot

[root@study ~]# uname -a
Linux study.centos.vbird 4.2.3-1.el7.centos.x86_64 #1 SMP Wed Oct 21 02:31:18 CST 2015 x8
x86_64 x86_64 GNU/Linux
```

这样就让我们的 CentOS 7.x 具有最新的核心啰！与核心官网相同版本咧～够帅气吧！

24.6 重点回顾

- 其实核心就是系统上面的一个文件而已，这个文件包含了驱动主机各项硬件的侦测程序与驱动模块；
- 上述的核心模块放置于：/lib/modules/$（uname -r）/kernel/
- “驱动程序开发”的工作上面来说，应该是属于硬件发展厂商的问题
- 一般的使用者，由于系统已经将核心编译的相当的适合一般使用者使用了，因此一般入门的使用者，基本上，不太需要编译核心
- 编译核心的一般目的：新功能的需求、原本的核心太过臃肿、与硬件搭配的稳定性、其他需求（如嵌入式系统）
- 编译核心前，最好先了解到您主机的硬件，以及主机的用途，才能选择好核心功能；
- 编译前若想要保持核心源代码的干净，可使用 make mrproper 来清除暂存盘与配置文件；
- 挑选核心功能与模块可用 make 配合：menuconfig, oldconfig, xconfig, gconfig 等等
- 核心功能挑选完毕后，一般常见的编译过程为：make bzImage, make modules
- 模块编译成功后的安装方式为： make modules_install
- 核心的安装过程中，需要移动 bzImage 文件、创建 initramfs 文件、重建 grub.cfg 等动作；
- 我们可以自行由硬件开发商之官网下载驱动程序来自行编译核心模块！

24.7 本章习题

（要看答案请将鼠标移动到“答：”下面的空白处，按下左键圈选空白处即可察看）

- 简单说明核心编译的步骤为何？

  1. 先下载核心源代码，可以从 http://www.kernel.org 或者是 distributions 的 SRPM 来着手；
  2. 以下以 Tarball 来处理，解开源代码到 /usr/src/kernels 目录下；
  3. 先进行旧数据删除的动作：“make mrproper”；
  4. 开始挑选核心功能，可以利用“make menuconfig”、“make oldconfig”、“make gconfig”等等；
  5. 清除过去的中间暂存盘数据：“make clean”
  6. 开始核心文件与核心模块的编译：“make bzImage”、“make modules”
  7. 开始核心模块的安装：“make modules_install”
  8. 开始核心文件的安装，可以使用的方式有：“make install”或者是通过手动的方式复制核心文件到 /boot/ 当中；
  9. 创建 initramfs 文件；
  10. 使用 grub2-mkconfig 修改 /boot/grub2/grub.cfg 文件；

- 如果你利用新编译的核心来操作系统，发现系统并不稳定，你想要移除这个自行编译的核心该如何处理？

  1. 重新开机，并使用旧的稳定的核心开机！
  2. 此时才可以将新版核心模块删除： rm -rf /lib/modules/3.10.89vbird
  3. 删除掉 /boot 里面的新核心： rm /boot/vmlinuz-3.10.89vbird /boot/initramfs-3.10.89vbird.img ...
  4. 重建 grub.cfg： grub2-mkconfig -o /boot/grub2/grub.cfg

24.8 参考资料与延伸阅读

- [1]通过在 /usr/src/kernels/linux-3.10.89 下面的 README 以及“ make help ”可以得到相当多的解释
- 核心编译的功能：可以用来测试 CPU 性能喔！因为 compile 非常耗系统资源！

2002/05/29：第一次完成 2003/02/11：重新编排与加入 FAQ 2004/06/11：原本的 2.4.xx 版本核心被移动到 此处 2005/11/15：原本的模块管理已经先移动到开机流程管理那一篇啰！ 2005/12/05：经过将近一个月，呵呵！终于给他整理出来这一篇了～真难得～ 2007/06/27：增加了 initrd 的简单说明，详细还是得看 loader 那一章。 2009/07/21：将基于 FC4 所撰写的文章移动到此处 2009/08/03：原本的 KDE/GNOME 使用的发动机写错了！ KDE 用 Qt ，而 GNOME 是用 Gtk ！非常感谢 Chua Tze An 兄提供的指正！ 2009/09/18：加入两个简单的题目，给大家思考一下而已。 2015/09/23：将基于 CentOS 5 的旧的版本移动到这里。